A COMPUTER WARGAME SIMULATION

FOR MICROCOMPUTERS

A Thesis

by

CHARLES EDMUND HARRISON III

Submitted to the College of Graduate Studies
Texas A&I University
in partial fulfillment of the requirements for the
degree of

MASTER OF SCIENCE

August 1986

Major Subject: Computer Science

A COMPUTER WARGAME SIMULATION

FOR MINICOMPUTERS

A Thesis

by

CHARLES EDMUND HARRISON III

Approved as to style and content by:

D. L. Moorehead

Donovan L. Moorehead Ph.D.
(Chairman of Commitee)

Yuan R. Wang

Yuan R. Wang Ph.D.
(Member)

David R. Cecil

David R. Cecil Ph.D.
(Member)

Yuan R. Wang

Yuan R. Wang Ph.D.
(Head of Department)

Robert O. Kirby

Robert O Kirby Ph.D.
(Member)

August 1986

ABSTRACT

A Computer Wargame Simulation for Microcomputers

August 1986

Charles Edmund Harrison III,

B.S.,United States Military Academy

Chairman of Advisory Committee: Dr. Donovan L. Moorehead

The computer wargame simulation is a low resolution model used to simulate brief periods of combat in limited geographical areas. This simulation is adapted for use on any IBM compatible microcomputer with 128,000 or more Bytes of memory. This model may be used for training, command post exercises, or analyzing differing tactics, force structures, and weapon systems. Due to the complexity of the problem and the limited memory available, three separate programs are utilized. The first program sets up the data base, the second performs preliminary calculations and allows the gamer to input specific situational data, and the third program actually calculates the results of the combat.

ACKNOWLEDGMENT

I would like to acknowledge the assistance given to me by Dr. Donovan L. Moorehead in completing this project. I would also like to acknowledge the United States Army Combined Arms Combat Development Activity (CACDA) for training me and giving me the opportunity to serve as a war gamer for two years. Finally, I would like to thank LTC Visente V. Trevino for allowing me the time and flexibility to complete this project as well as a Master's of Science Degree.

TABLE OF CONTENTS

DISCLAIMER

The opinions expressed herein are strictly those of the author and do not necessarily reflect the official positions or doctrine of the United States Army, Texas A&I University, or the Department of Military Science. All of the data contained in this report or used in the simulation has been obtained from unclassified sources and the results of the simulation are not guaranteed to be an accurate prediction of an actual combat situation.

CHAPTER I

INTRODUCTION

1.1 PURPOSE. This is a low resolution model which may be utilized to assist gamers simulating combat. It may be utilized for training, command post exercises, and comparison of various tactics, force structures, and weapons.

1.2 QUALIFICATIONS OF GAMERS. The gamers do not need to have any extensive training in computers or the Pascal language. The computer algorithms merely assist the gamer by performing some record keeping and attrition calculation functions. The gamer must be well trained in basic military tactics and force structures.

1.3 PREPARATION. Before the gamer can begin utilizing the attrition program, he must first load the appropriate data into the data base. Program Build is utilized for that purpose and it is written simply enough so that any gamer should not have difficulty in inputting these characteristics into the data base.

Every unit in the military has a Table of Organization and Equipment or a similar authorization document which describes in detail the types and quantity of each item of equipment found in that type unit.[*1] The gamer will use that information to

---

Journal of the Association for Computing Machinery

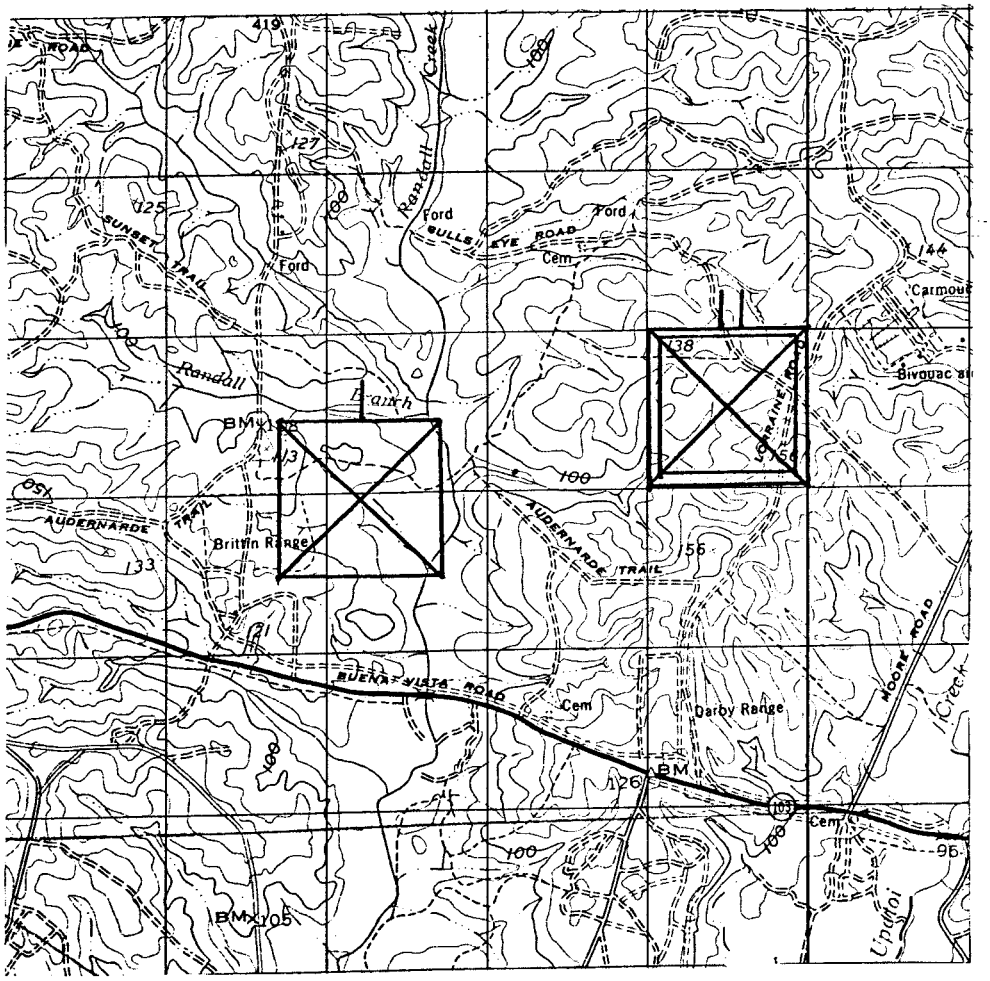
* Superscripted numbers refer to ENDNOTES found on page 113

input the names, types of equipment, and quantity of each into the data base using program Build.

The gamers will then place unit symbols on a 1: 50,000 scale standard military map.[2] Military maps are ideally suited for war gaming because of the presence of lines of equal altitude known as "contour lines". By viewing the pattern of the contour lines, the gamer can determine the general pattern of terrain.[3]

FIGURE 1

UNIT SYMBOLS LOCATED ON A MILITARY MAP



The gamers must then establish geographical sectors within which every unit is within a reasonable engagement range of every opposing unit inside of that sector. In addition, the artillery units that are to be utilized against the elements in that sector must be identified. In a large operation the battle may consist of numerous sectors.

1.4 COMPUTATION OF ATTRITION. The gamer will utilize Program Prep to load specific environmental and situational factors into the data base. In addition, the gamer will select the number of attack helicopters that will be utilized in every sector.

The gamer will use Program Game to actually calculate the attrition for that given sector. The gamer will have the options of running any one of four attrition routines in any order or as many times as desired.

1.5 NEXT ITERATION. Following the completion of gaming all of the sectors, the gamers will rearray their forces on the maps, confirm or change the location of the sectors, adjust the input parameters, and proceed to calculate the new losses for the sectors. This process will continue until the exercise is completed.

CHAPTER II

METHODOLOGY AND ASSUMPTIONS

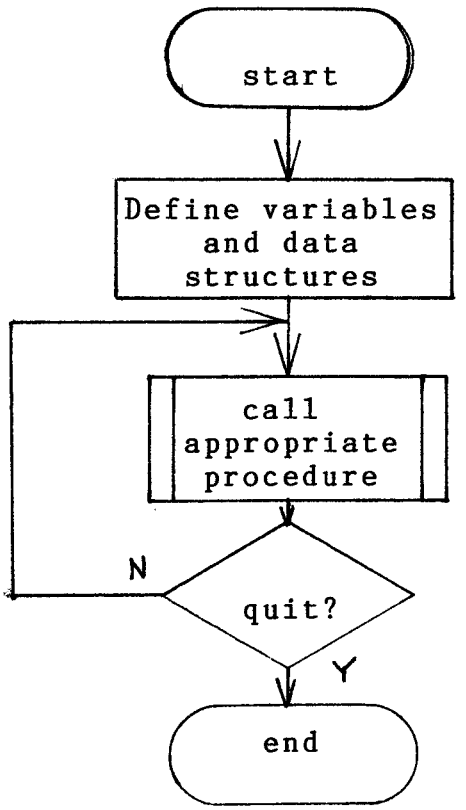
2.1 PROGRAM BUILD. The purpose of Program Build is to write the information to the data base so that the other two programs can access the information later. The program is written so that an untrained operator can use it. This program, like the other two, is written in Pascal, utilizing the Turbo Pascal Compiler.[4]

The record structure is used extensively throughout all three programs. File variables are assigned to the appropriate records and the program will write the information to the diskette.

2.1.1 MAIN PROCEDURE. As indicated by the flow diagram on the next page, the main procedure is simple. The computer will display a message to the user asking him to enter a number from 1 to 7 indicating the action which the gamer wishes to take. A Pascal case statement will then make a call to the appropriate procedure. In addition, the computer will print an error message if the user enters an incorrect number or a character. The user will simply make as many procedure calls as necessary.

FIGURE 2

MAIN PROCEDURE FOR PROGRAM BUILD

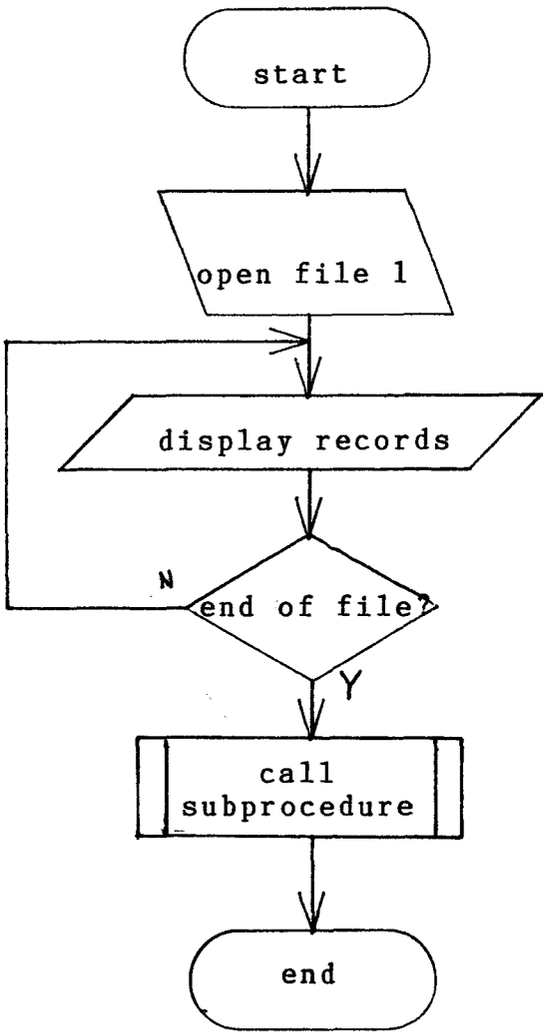


2.1.2 PROCEDURE WUPDATE (Weapons Update Procedure). Most of the procedures used in this program are relatively standardized in that they use four nearly identical procedures. Procedure Insert will write the new record into the file. The user will call one of three other procedures which will, in turn, call Procedure Insert. Procedure Add will write a new record to the end of the file. Procedure Update will sequentially review each record and give the gamer an opportunity to change the record until the end of the file is reached. Procedure Build will open the file, erase any previous data, and write new records until the gamer is finished. For the Wupdate Procedure only, these subprocedures will be described in detail.

As shown on the following page, the weapons update procedure will simply display the current contents of the file and allow the gamer to invoke the proper subprocedures.

FIGURE 3

MAIN WEAPONS UPDATE PROCEDURE

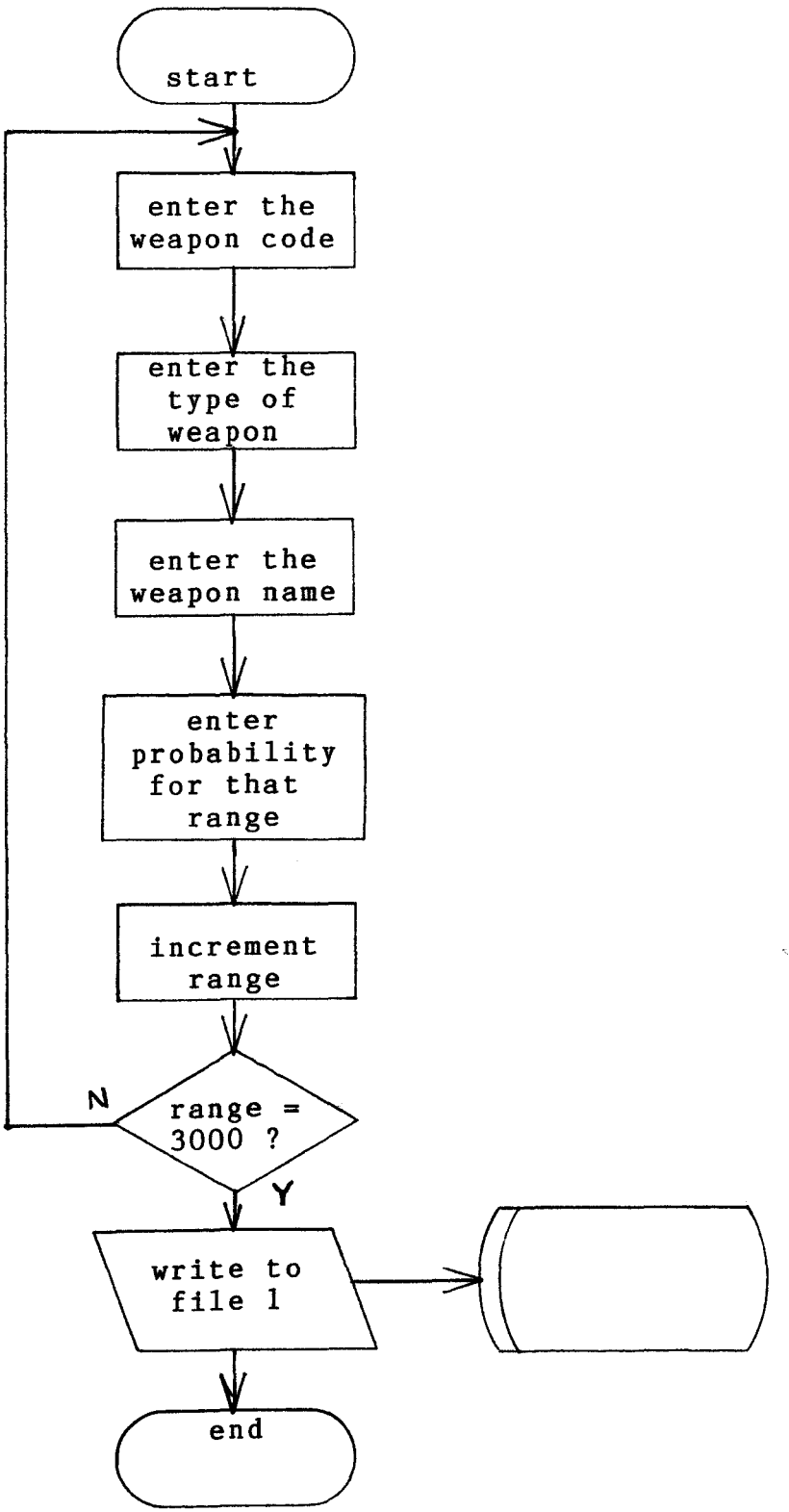


As noted before, Procedure Insert will be called by the other three procedures as needed. The routine will print prompts to the user to input the proper appropriate response. At the end of the record the file will be written to the diskette. The weapon code is an integer used to identify a particular weapon system. The weapon type is another integer utilized to designate a system as a Tank, Armored Personnel

Carrier, an Armored Personnel Carrier with an Antitank Capability, an Anti Tank Guided Missile, or an Air Defense System. This distinction will be very important when Program Game is run. The probabilities are the probabilities that that particular weapon system will kill a target. The probabilities are defined for every 500 meters similar to those used in the Dunn-Kempf manual war game.[5]

FIGURE 4

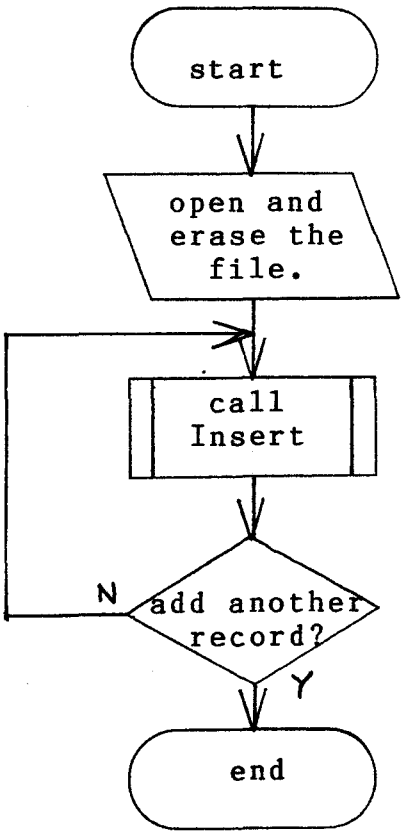
PROCEDURE INSERT FOR THE WEAPONS UPDATE PROCEDURE



Procedure Build (not to be confused with Program Build) will erase the old file, open it, and allow the user to call subprocedure Insert as many times as necessary.

FIGURE 5

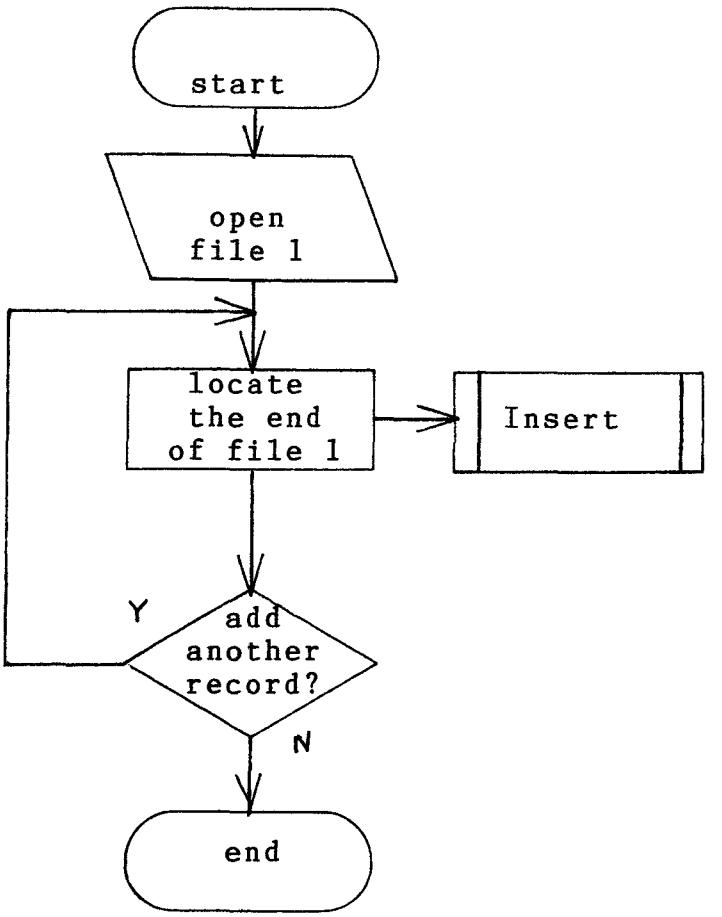
PROCEDURE BUILD FOR THE WEAPONS UPDATE PROCEDURE



Procedure Add will open the file and locate the end of the file. Procedure Insert will then be called as many times as necessary.

FIGURE 6

PROCEDURE ADD FOR WEAPONS UPDATE PROCEDURE

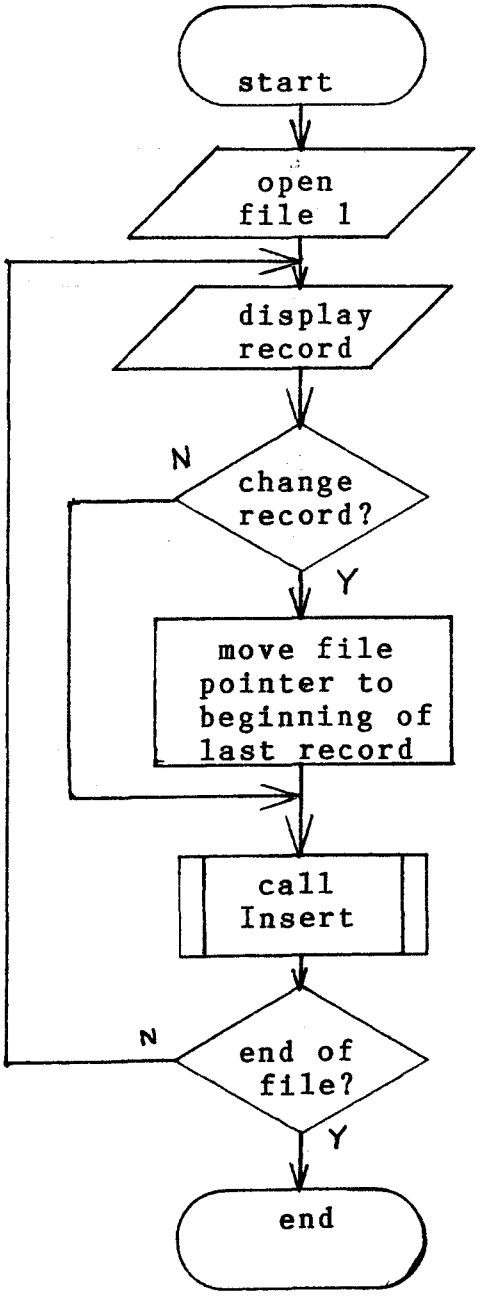


Procedure Update is slightly more complex than the others in that it will allow the gamer to selectively review and correct each record in the file if needed. This saves a great deal of time when using a particularly long file. The procedure will first open the file at the beginning, display the the contents of the next record, and ask the gamer if he wishes to change the contents of that record. If he does wish to make a change, the file pointer is placed back to the beginning of the record just read and

Procedure Insert is called in order to input a new record into the file. This procedure will continue until the end-of-file marker is reached.

FIGURE 7

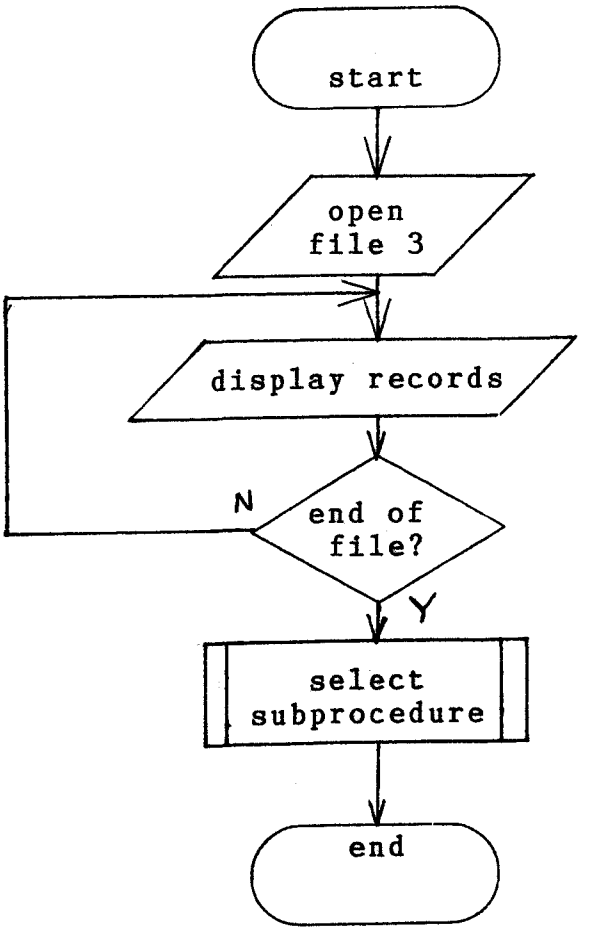
PROCEDURE UPDATE FOR WEAPONS UPDATE PROCEDURE



2.1.3 PROCEDURE UUPDATE (Unit Update Procedure). This procedure will use the same four subprocedures to build, update, or add records to file 3, which contains the unit information. The contents of the unit file are displayed and then the gamer is allowed the same options as with the Procedure Wupdate. The difference is found in the record structure which contains the name of the unit, the number of weapons in the unit, and the weapon code and quantity for each weapon in the unit.

FIGURE 8

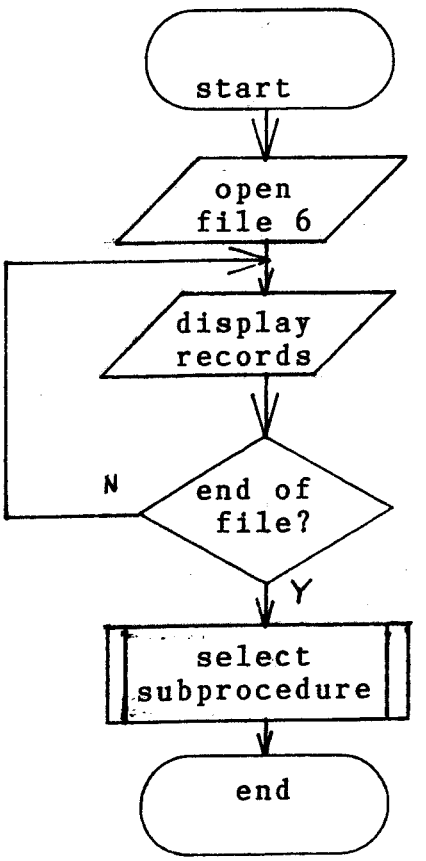
UNIT UPDATE PROCEDURE FOR PROGRAM BUILD



2.1.4 PROCEDURE ARTY (Artillery Update Procedure). This procedure will use the same four subprocedures to build, update, or add records to file 6, which contains the unit information. The contents of the artillery file are displayed and the gamer is allowed the same options as with the weapons update procedure. The difference is found in the record structure which contains the name of the artillery piece, the size of the system (small mortar, light artillery, medium artillery, or heavy artillery), the size of a one day supply of ammunition, and the weapon code. Weapon codes from 1 to 9 are reserved for blue forces and 10 to 20 are reserved for red forces.

FIGURE 9

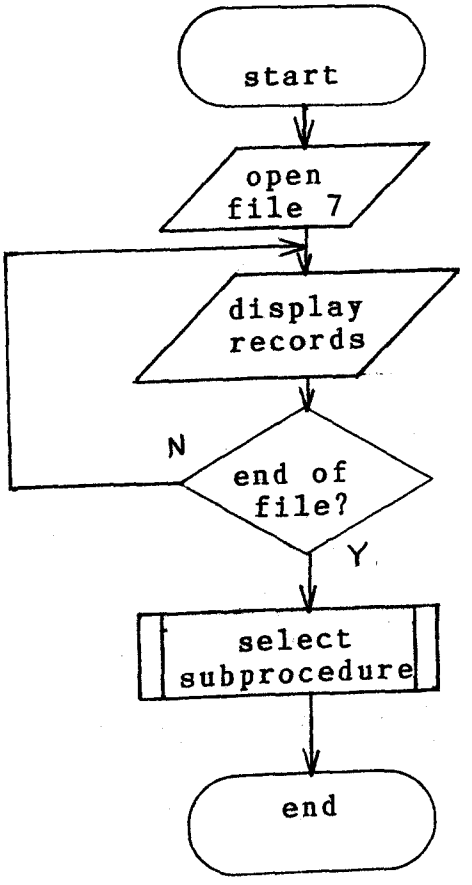
ARTILLERY UPDATE PROCEDURE FOR PROGRAM BUILD



2.1.5 PROCEDURE ARTUNIT (Artillery Unit Update Procedure) This procedure will use the same four subprocedures to build, update, or add records to file 7, which contains the unit information. The contents of the artillery unit file are displayed and the gamer is allowed the same options as with the weapons update procedure. The difference is found in the record structure which contains the name of the unit, the weapon code of the Artillery piece used by that unit and the number of weapons in the unit. Unlike other units, Artillery Batteries normally contain only one primary weapon.

FIGURE 10

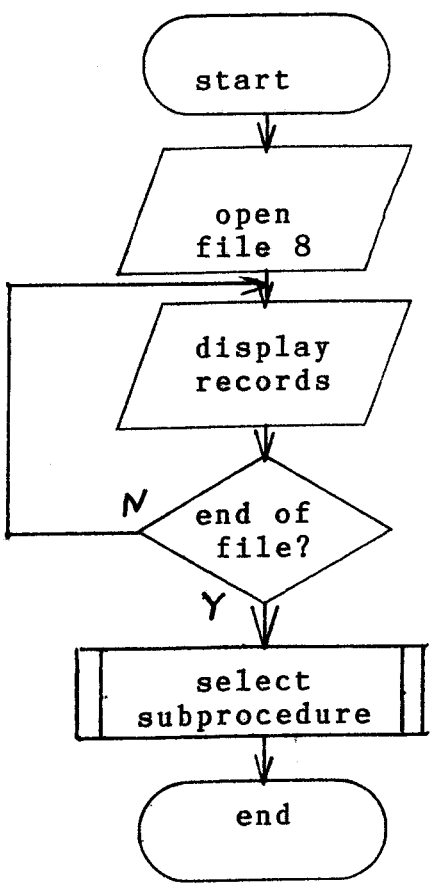
PROCEDURE ARTUNIT FOR PROGRAM BUILD



2.1.6 PROCEDURE AIR (Attack helicopter Update Procedure). This procedure will use the same four subprocedures to build, update, or add records to file 8, which contains the attack helicopter information. The contents of the unit file are displayed and the gamer is allowed the same options as with the weapons update procedure. The difference is found in the record structure which contains the name of the helicopter, the maximum range of the helicopter, the weapon code, the probability of a kill against an armor target, and a boolean variable which is true for side blue and false for side red.

FIGURE 11

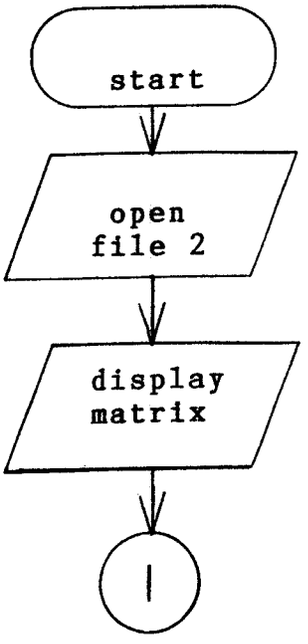
HELICOPTER UPDATE PROCEDURE FOR PROGRAM BUILD

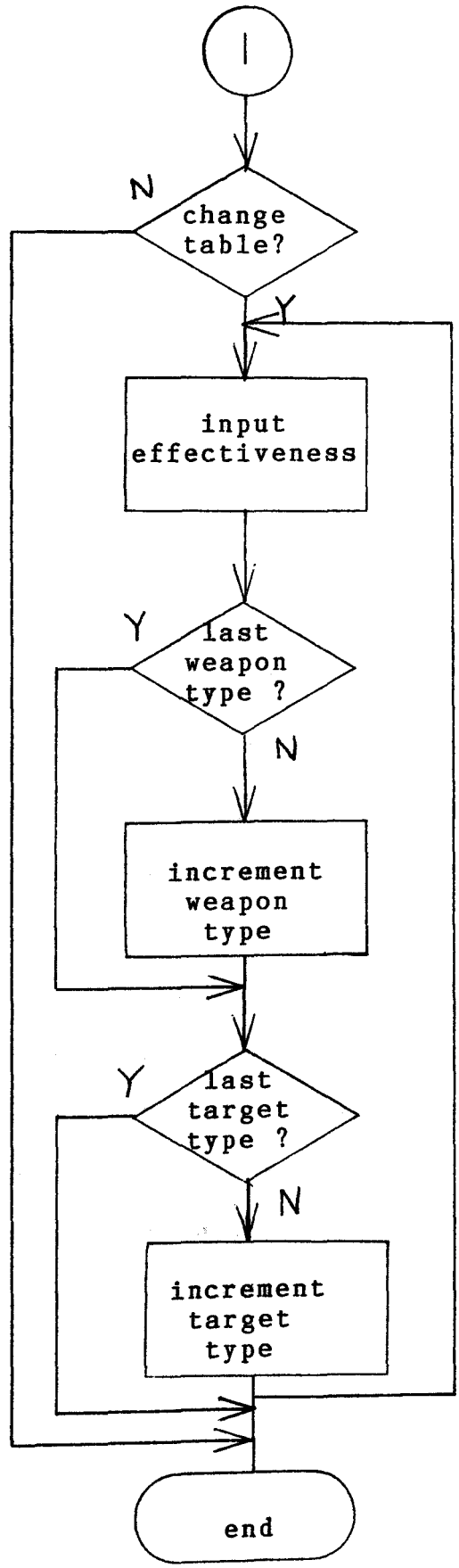


2.1.7 PROCEDURE MATRIX (Artillery effectiveness matrix update procedure). This procedure is considerably different from the other Procedures. The purpose of the procedure is to allow the gamer to input effectiveness tables which correspond to the size of the artillery piece and the vulnerability of the target. The values used by the author correspond roughly to the probabilities used by the Dunn-Kempf war game.[6] First, file 2 is displayed to the gamer. Then, the gamer is prompted to indicate whether or not the table should be changed.The computer will assist the gamer by prompting the weapon size and target type for each value that is entered. After all 20 values are entered, the procedure will end.

FIGURE 12

ARTILLERY EFFECTIVENESS UPDATE PROCEDURE FOR PROGRAM BUILD




change table?
N
Y
input effectiveness
last weapon type ?
Y
N
increment weapon type
last target type ?
Y
N
increment target type
end

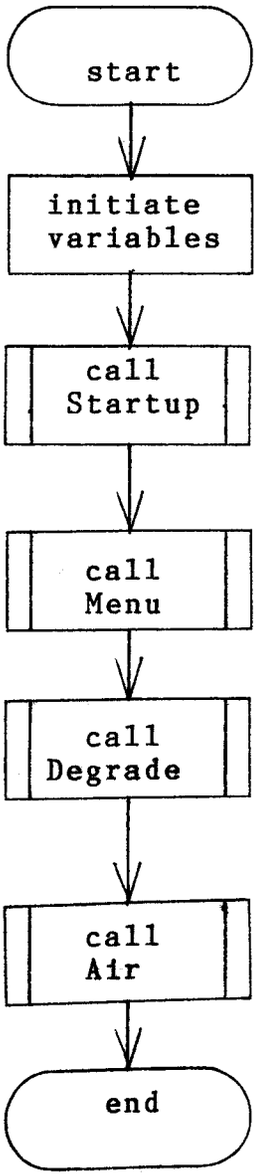
2.2 PROGRAM PREP. The purpose of Program Prep is to read in the information from the data base, to allow the gamer to input situational and environmental factors and schedule helicopter sorties, and to perform preliminary calculations which will be required by Program Game.

In addition to the record structures used previously, this routine utilizes linked lists with pointer variables. This structure has several advantages for the user in that the amount of storage required is dynamically allocated requiring only the amount of memory needed to be set aside. In addition, the reaction time of the computer is much faster since this reduced memory requirement allows the entire record to be kept in memory rather than reading files from the diskette in order to make comparisons or define values for variables, etc.. (NOTE: After the author changed some of the algorithms from utilizing statically defined arrays and searching for particular values from the diskettes the response time improved by approximately a factor of 10.)

2.2.1 MAIN PROGRAM. The main program in program Prep is trivial. The program merely initiates some of the variables and calls three procedures: Startup, Menu, and Air in turn. (Procedure menu will call a fourth procedure, Degrade.)

FIGURE 13

MAIN PROCEDURE FOR PROGRAM PREP

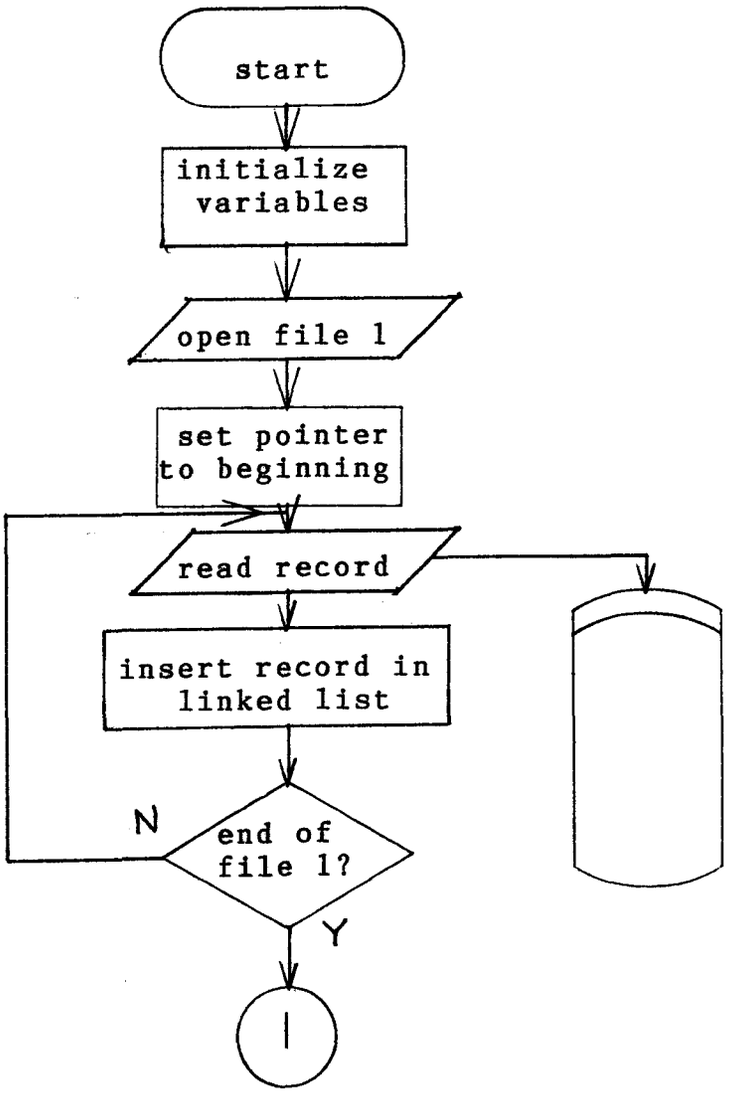


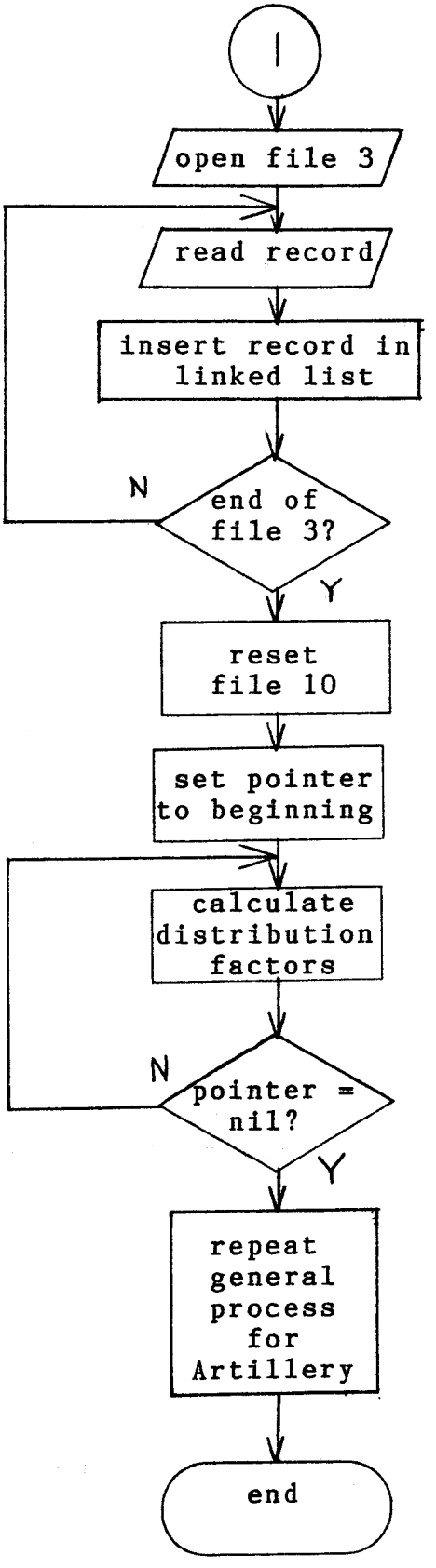
2.2.2 PROCEDURE STARTUP. This procedure is considerably more complex than the main procedure. The records are read in from file 1. Space is then allocated for the linked list to be stored in memory. The quantity of the weapons is initially set to zero. All weapons with code numbers less than 50 are blue weapons and a boolean variable known as "side" is given a value of true. All other weapons are given a value of false. All other weapon characteristics are directly read in from the records in file 1. The pointer variable is then incremented to the next entry. This process is repeated until the end-of-file marker is reached. The unit file(3) is then opened and the first record is read. The weapons linked list is searched until a record equal to that weapon code is found. At this point the quantity is incremented by the number of that type of weapon. This step is repeated until the last weapon type for that record is located and the corresponding quantity added to the value presently in the linked list. This process will then continue until the last unit record has been read. The linked list is then searched again and the total number of blue and red weapons is summed. A distribution factor is than calculated for each weapon type by dividing the quantity of that weapon by the total number of weapons for that side. This distribution factor will be used in Program Game to calculate losses. That distribution factor is loaded into the linked list at this time. File 10 is then opened and erased. The linked list structure is then written into file 10 (except for the pointer

variables which cannot be saved to the diskette in TURBO Pascal.[7]

FIGURE 14

PROCEDURE STARTUP FOR PROGRAM PREP




open file 3
read record
insert record in linked list
N
end of file 3?
Y
reset file 10
set pointer to beginning
calculate distribution factors
N
pointer = nil?
Y
repeat general process for Artillery
end

2.2.3 PROCEDURE MENU. Procedure Menu is a relatively simple procedure which will prompt the gamer to input parameters describing the particular situation to a record which will be later written to the diskette. Procedure Menu will then call Procedure Degrade.

2.2.4 PROCEDURE DEGRADE. Procedure Degrade will take the parameters input by the gamer in procedure Menu and evaluate various degradation factors which will dampen the effects of weapon systems utilized in Program Game. These degradation factors are discussed in detail below since they represent important assumptions which will be critical to the simulation. It is important to realize that without these factors the algorithms will treat reality as a perfectly flat pool table where every weapon system can acquire, engage, and destroy every opposing weapon system in the sector unaffected by the numerous factors which occur in real combat.

ASSUMPTION 1:

In " average terrain" the probability of having line of sight from the target to the firer is assumed to be normally distributed with a mean of 1500 meters and a standard deviation of 1000 meters.[8]

DISCUSSION:

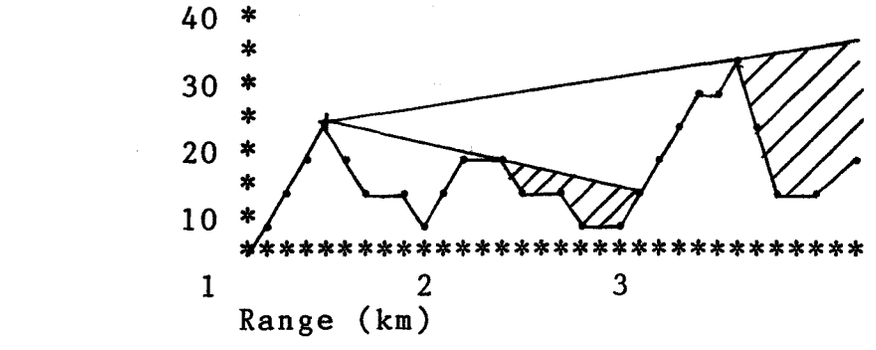
This is a rough approximation for 2 reasons: 1, there is no readily available documentation if it even exists; and 2, the mean and standard deviation would vary from any given piece of terrain to any other given piece of terrain. High

resolution models will often handle this problem by digitizing the terrain and actually calculating the line of sight from a firer to a target at every possible location. This is, of course, well beyond the scope of this model.

FIGURE 15

LINE OF SIGHT EXAMPLE

+ Firer /// dead space



In the above illustration, the firer cannot engage targets in the slashed area with direct fire.

ASSUMPTION 2 :

The roughness of the terrain will add a degree of protection to potential targets on both sides, due to the provision of cover and concealment. Roughly, the hillier the terrain, the more likely there is to be large rocks, trees, and ditches which can provide protection against ballistic projectiles or concealment from enemy gunners. Once again,

there is no iron-clad scientific basis for this assumption other than experience. For purposes of this simulation this factor will reduce the number of effective engagements reaching the target. The estimates used for this degradation factor are:

Open terrain	0 %
Rolling Hills	25 %
Hilly	50 %

ASSUMPTION 3:

Defensive Preparations will provide an advantage to the defender in that they will have better cover than would normally be provided by natural terrain. Although this factor has been generally accepted throughout history, it is difficult to quantify. For the purpose of this simulation the Dunn-Kempf factors are utilized:[10]

Prepared Defensive Positions	70 %
Hasty Defensive Positions	30 %
No Preparations (Meeting Engagement)	0 %

DISCUSSION:

Normally a unit with sufficient warning of an attack will make extensive preparations such as Anti-Tank ditches, dug in fighting positions and obstacles. A unit with less time will look for hull or turret defilade positions and attempt to make minor improvements to the protection provided by nature.

2.2.5 PROCEDURE AIR. This procedure will review the records for each type of helicopter loaded in the data base and prompt the gamer to input the number of the sorties to be flown by that type of helicopter against the opposing armor. First, the algorithm will output the a record from file 8. Next, gamer will be prompted to input the number of sorties to be flown by that particular helicopter. A zero is entered to indicate that no sorties are flown. The record with the number of sorties included will then be written to file 12. This process is continued until the end of file 8 is reached.

2.3 PROGRAM GAME. The purpose of this program is to simulate the combat results of an actual engagement. The inputs are the files created by Program Prep. The output is a list of systems killed by a routine, the cumulative kills of the game, and the list of remaining systems.

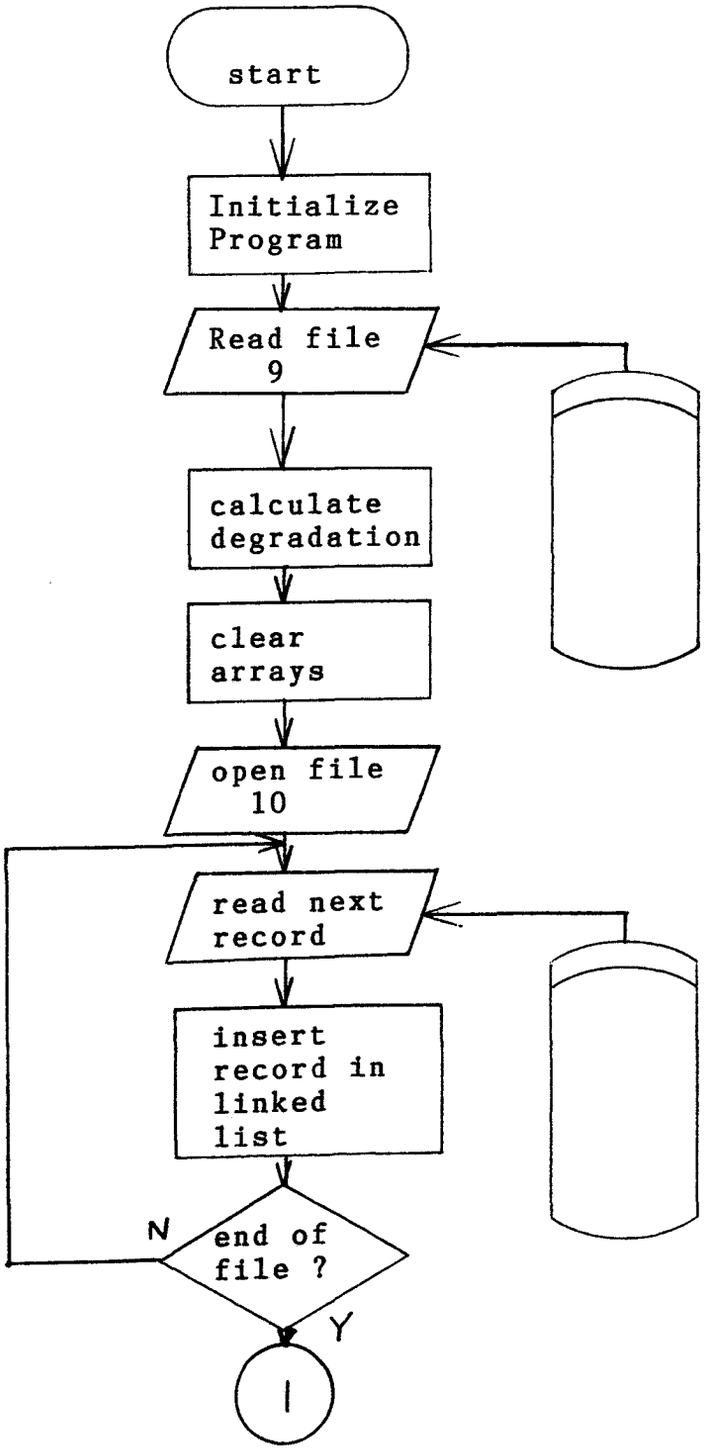
The data structures utilized by this program are the record and linked list structures.

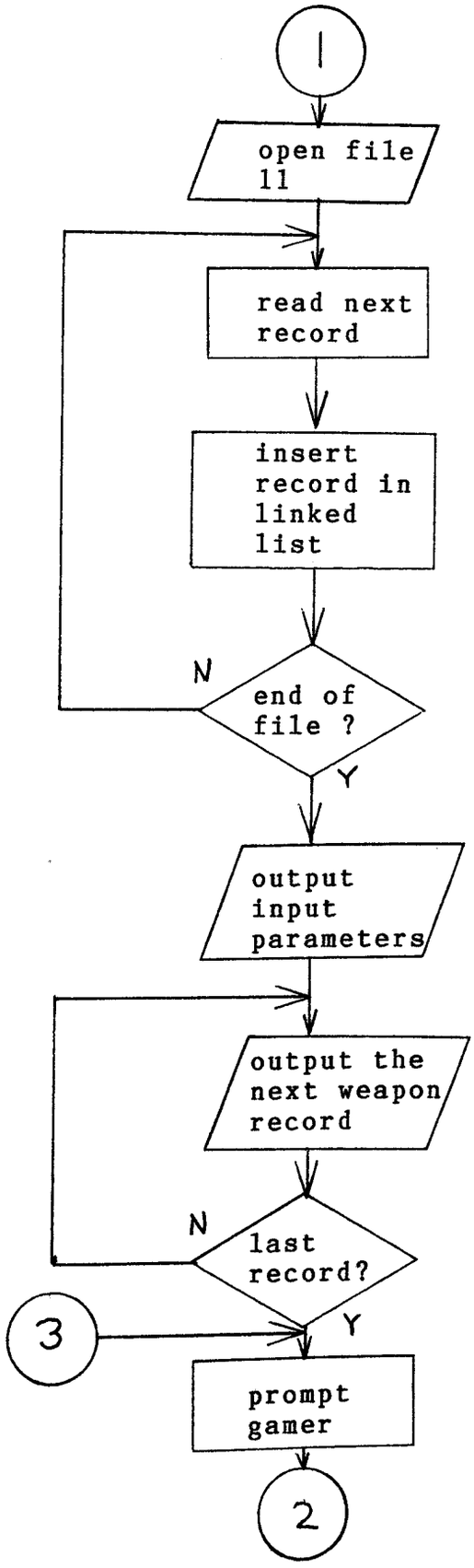
2.3.1 MAIN PROGRAM. The variables are defined and initialized. File 9 is read from diskette which contains the input parameters from program Prep. The degradation variable is then defined in terms of the input parameters. Several arrays which will later store information relating to the number of kills of each type weapon, etc. will be cleared. File 10 is then opened and the weapons linked list is built one record at a time until the end of file marker is reached. The same procedure is used with file 11 to build the artillery linked list. The list of input parameters is then printed out on the screen. The list of weapons is then output using the linked list. The gamer is then prompted to select one of three attrition routines, Artillery, Air, or Armor. A case statement will then initiate a subroutine call to the proper procedure. Upon return to the main procedure, the cumulative list of kills and the number of weapons remaining will be output. Procedure distribute is then called to recalculate the percentage of the total force which that type of weapon system consists of. Finally, a check is nade to determine whether or not the gamer is finished. If a false reading is detected, the gamer will be prompted again to

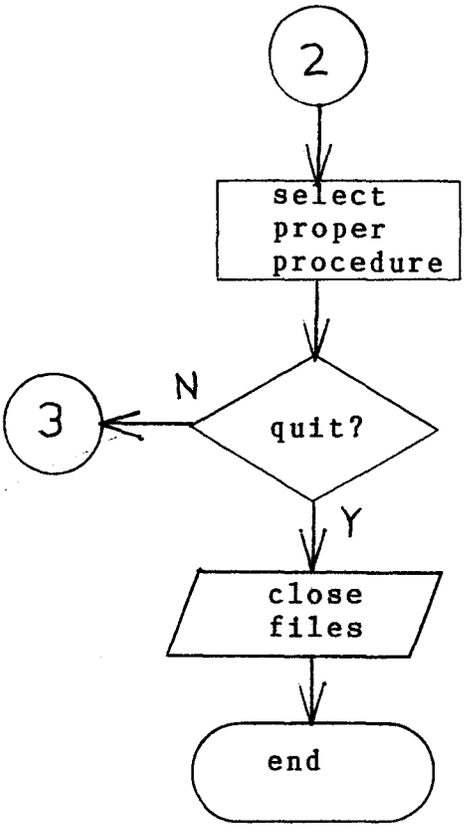
select a procedure call, and the process repeats until the gamer decides to "quit".

FIGURE 16

MAIN PROCEDURE FOR PROGRAM GAME



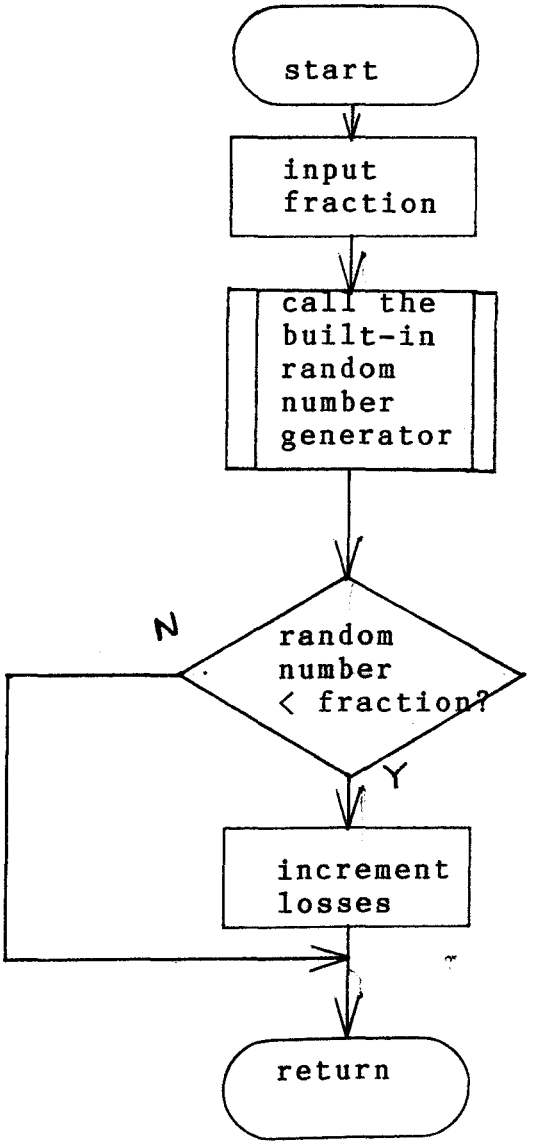

1
open file 11
read next record
insert record in linked list
N
end of file ?
Y
output input parameters
output the next weapon record
N
last record?
Y
3
prompt gamer
2



2.3.2 PROCEDURE RANDOM. This procedure is a simple procedure used to determine whether or not a weapon system has been killed. The game assumes that every engagement is a Bernoulli trial. The expected number of effective kills then is equal to nP with n being equal to the number of rounds fired and P being equal to the kill probability.[11] However, since it is difficult to kill half of a weapon, a random number generator is called in order to determine whether or not the fractional portion of nP consists of a kill. If the random number is smaller than the fractional part then an extra kill is assessed.[12] This is similar to the Dunn-Kempf War Game.

FIGURE 17

PROCEDURE RANDOM FOR PROGRAM GAME



2.3.3 PROCEDURE DISTRIBUTE. This procedure will recalculate the percentage of the total force that the quantity of a particular weapon system consists of.

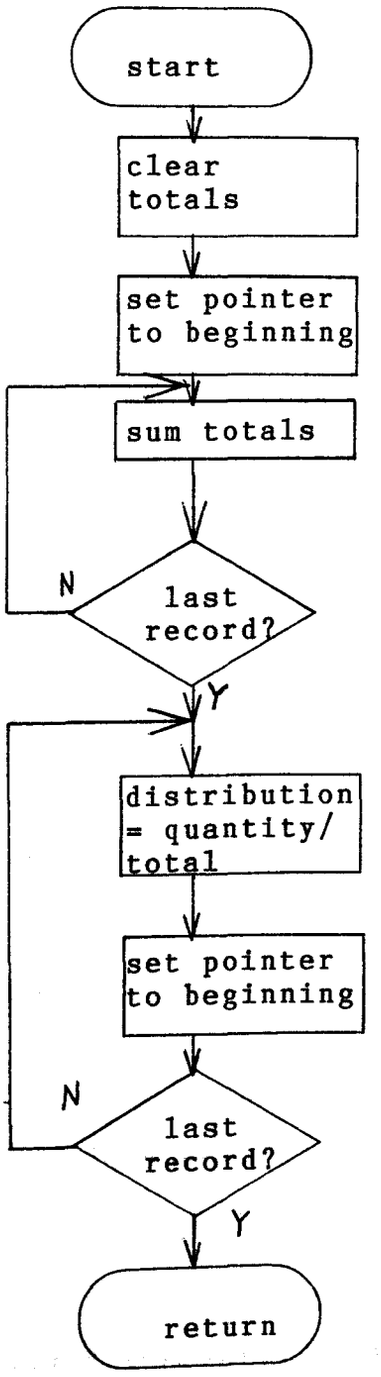
ASSUMPTION 4: Every type of weapon system is equally likely to be engaged as any other type of weapon.

DISCUSSION: Obviously, in actual combat the most serious threat is engaged by any given gunner given that he is able to acquire and fire at the target. However, considering the resolution of the model and the large number of independent engagements occurring simultaneously, it is a reasonable and probably necessary approximation.

Procedure Distribute will begin by setting the sum of red and blue systems equal to zero. Then the total quantity of blue and red systems is summed using the linked list. Finally, the distribution factor for each weapon is calculated by dividing the current quantities by the appropriate total.

FIGURE 18

PROCEDURE DISTRIBUTE FOR PROGRAM GAME



2.3.4 PROCEDURE APPORTION. This procedure will apportion the number of weapon kills to the opposing targets. The number of kills from an attack helicopter, artillery piece, or armored system is equal to the number of rounds fired multiplied by the probabilities and other degradation factors. This procedure will not assess a kill unless the firer and target are from different sides. The procedure will read each record one at a time from the weapons linked list and assess losses to each system.

ASSUMPTION 5:
The probability of detecting a target given that line of sight exists is inversely proportional to the range.

DISCUSSION:
There are no unclassified studies of this type. Any actual probability would be defined for every type of weapons system as a function of range. This program assumes a straight line relationship between the probability of detection and range. Once again, it is more realistic to include a rough approximation for this factor than to ignore it.

The probability of detection is defined to be equal to: (1-range)/ 5000. This would not be defined for a range greater than 5000 meters, however the probabilities are not defined for more than 5000 meters for any of the systems loaded into the data base, and the probabilities at 3000 meters approach zero.

ASSUMPTION 6:

Blue is the defender.

DISCUSSION:

Most war games assume that the Blue forces, normally representing the United States Forces, are defending against an attacking aggressor. In this simulation any advantages that are derived from a defensive posture will only assist whichever side is loaded as Blue. In the case of a US attack, the US forces would have to be loaded into the data base as Red forces.

The defensive preparation factors will only assist the defender. The number of effective rounds is multiplied by the distribution factor and the appropriate degradation factor. The game will then check a flag to see if the effective rounds came from the artillery procedure. If so, than the degradation factors are adjusted since artillery fires are not subject to the same line of sight and detection constraints as direct fire weapons. The value of the flag and the weapon type of the target locate the value of the effectiveness variable from the two dimensional matrix. The effective number of artillery kills is multiplied by that value. Next, the number of kills is determined using the Np and Procedure Random process described earlier. The code number of the target is used as the index for the kill array and the number of kills for the target is

incremented at that position. A statement will then check to see if the last weapon has already been killed and prevent the algorithm from killing weapons that no longer exist. The process is repeated until the end of the linked list is reached.

2.3.5.PROCEDURE ARTILLERY. The purpose of this procedure is to calculate the number of weapons systems destroyed by indirect fire.

This procedure will first clear the heading information. Then, the Artillery loss and kill arrays will be cleared. File 2 is read in with the effectiveness matrix. The gamer will then be prompted to select the firing rate in terms of the percentage of one day's ammunition consumed in a one hour period. The percentage of fires directed against enemy artillery is then selected. Procedure Print is then called to print out the list of artillery weapons systems remaining. The artillery linked list is reset to the beginning and the first record is then used to calculate the total number of shells fired by the following formula:

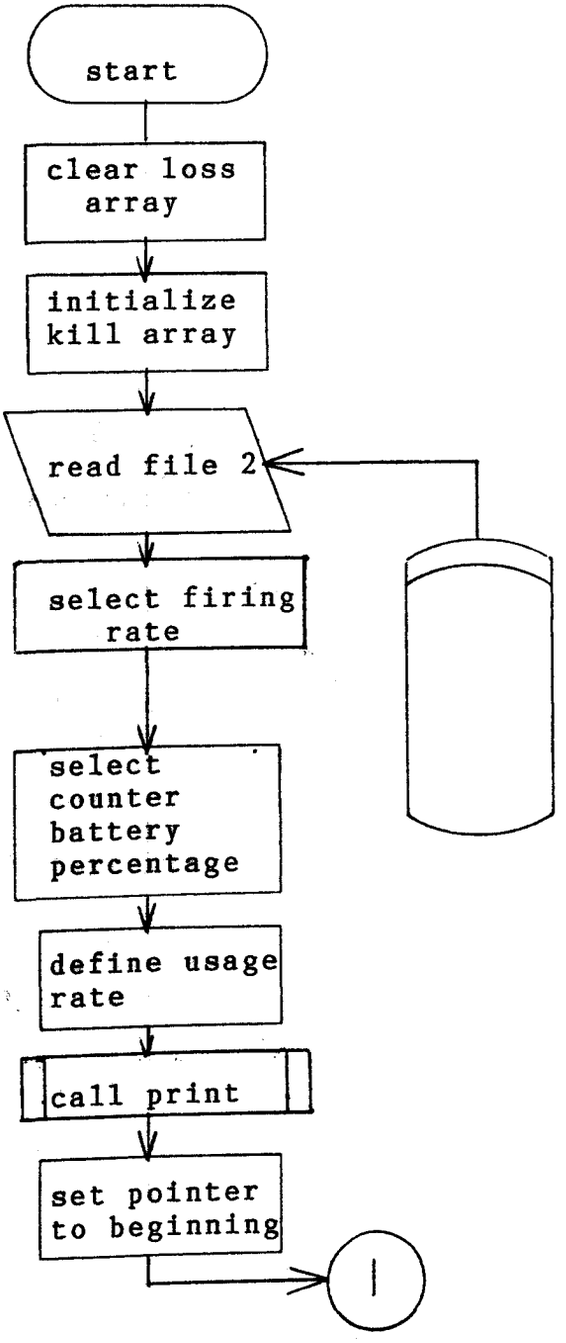
Total shells(artillery piece) = total supply X the time(minutes)/60 X the fraction of shells fired in 1 hour x the number of weapons left.

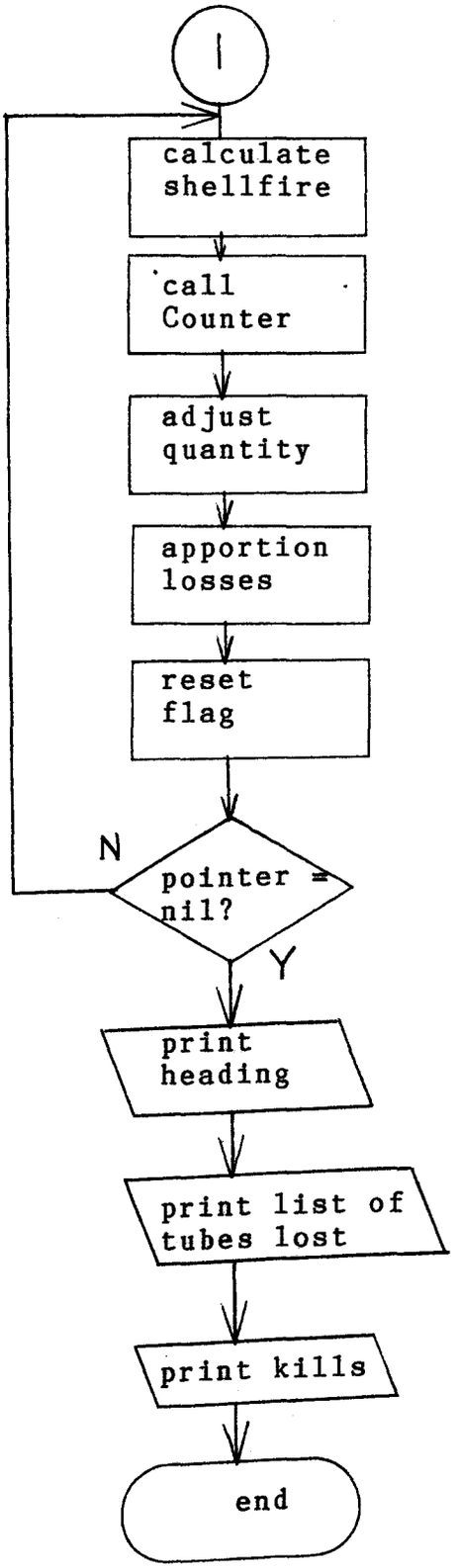
The side is defined from the record and the size of the firing piece is then read in. The number of shells is multiplied by the the percentage of fires directed at enemy artillery. Procedure Counter is then called to destroy some of the enemy artillery. The number of tubes remaining is then adjusted. The algorithm will not allow the number of tubes remaining to become less than zero. The remaining shells become the argument for procedure Apport which was discussed previously. The list of artillery tubes remaining is printed

out, Procedure Print is called to list the number of cannons remaining. Finally, the number of targets destroyed by artillery is printed out.

FIGURE 19

PROCEDURE ARTILLERY FOR PROGRAM GAME

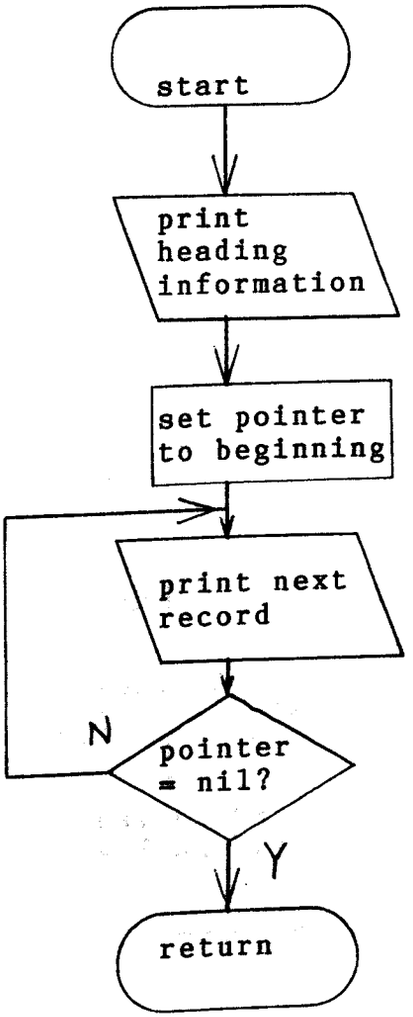



1
calculate shellfire
call Counter
adjust quantity
apportion losses
reset flag
N
pointer = nil?
Y
print heading
print list of tubes lost
print kills
end

Procedure Print is a simple procedure internal to Procedure Artillery which will print out the weapon code and quantity of each type of artillery remaining in the game. First, headings are printed out, the pointer variable is set to the beginning of the artillery linked list, and the quantity and weapon code of each record is printed out until the end of the list is reached. Procedure Artillery is then returned to.

FIGURE 20

PROCEDURE PRINT FOR PROCEDURE ARTILLERY



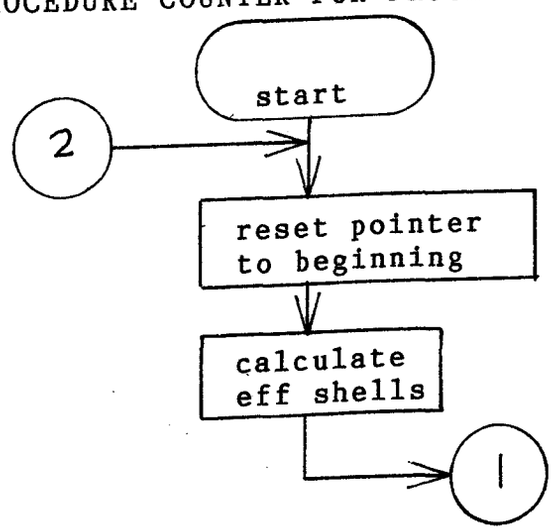
Procedure Counter is called by Procedure Artillery to calculate counterbattery attrition. The artillery pointer is reset to the beginning and the shells fired will only be used against opposing targets. The value of the effectiveness matrix will be defined in terms of the artillery type and target type. The effective number of shells fired at a particular opposing artillery piece is defined by the following formula:

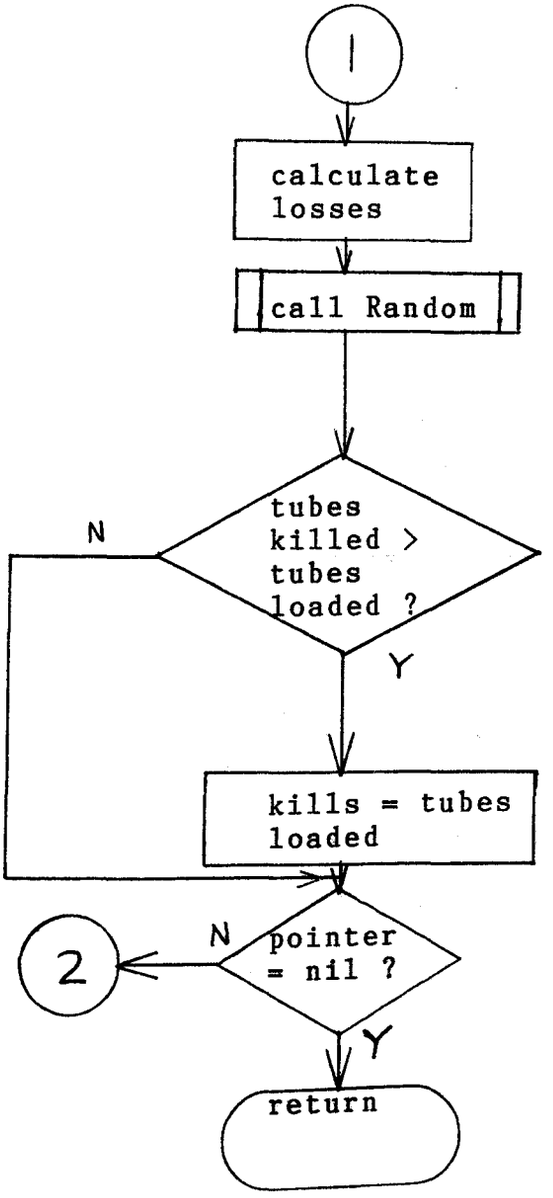
effective shells =shells X effectiveness(artillery size, target type) x distribution factor

Np tubes are killed and Procedure Random is called to evaluate fractional losses. The algorithm will not allow more tubes to be killed than were originally loaded. The process is continued until the end of the list is reached.

FIGURE 21

PROCEDURE COUNTER FOR PROCEDURE ARTY




1
calculate losses
call Random
tubes killed > tubes loaded ?
N
Y
kills = tubes loaded
pointer = nil ?
N
2
Y
return

2.3.6. PROCEDURE AIR. This procedure will calculate losses to and from attack helicopters.

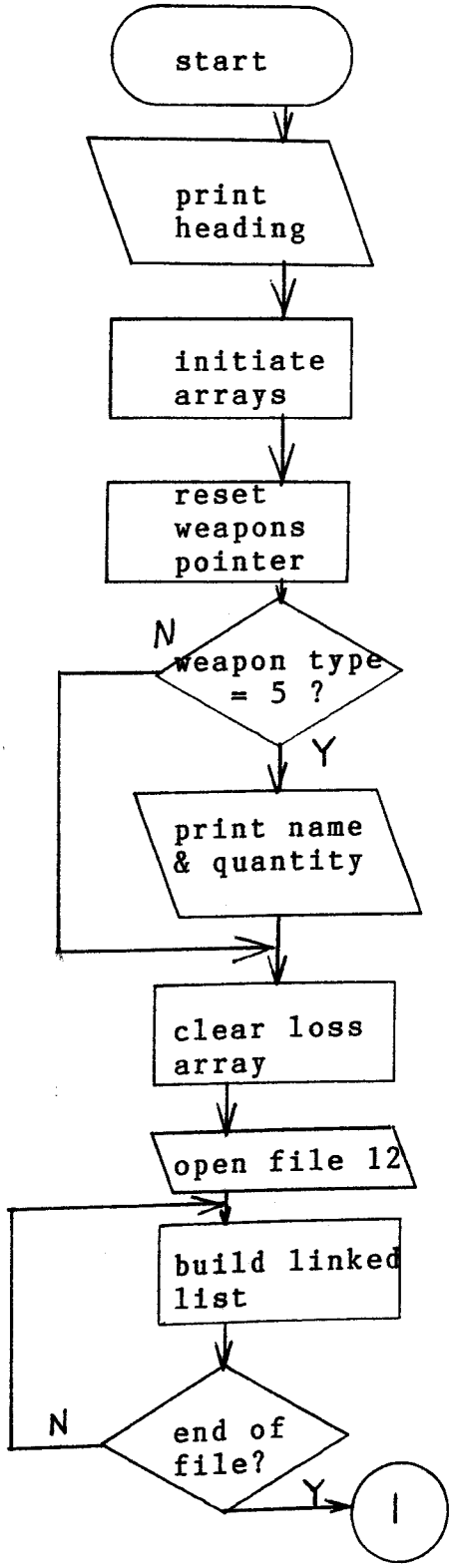
After printing heading information and initializing variables a list of air defense systems is printed out. The helicopter linked list is then built from file 12. The pointer is set to the beginning and the number of effective missiles fired is calculated for each type of helicopter until the end of the list is reached. Each helicopter will fire half of its missiles before being subjected to air defense fires. The remaining helicopters will then be able to fire the remaining missiles. The number of effective missiles is calculated by the formula :

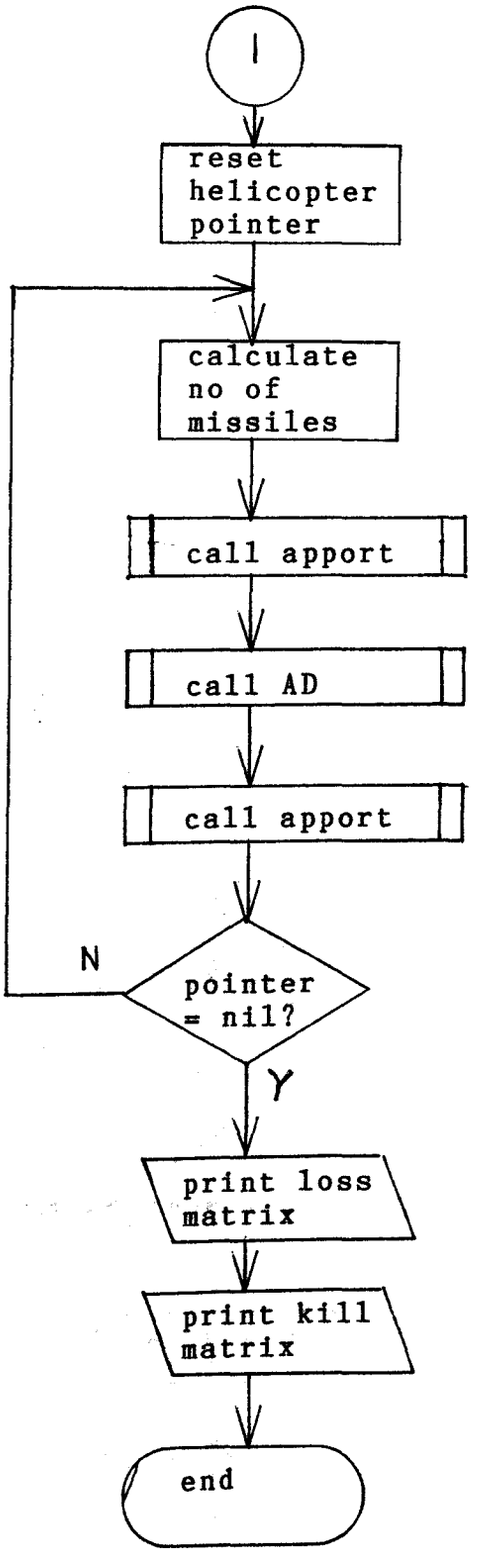
eff missiles = 4 x no of sorties x the kill probability(range).

Procedure Apport will then calculate the ground systems killed. Procedure AD will calculate the number of helicopters killed, and than Procedure Apport will be called again if there are any helicopters left. The number of helicopters killed and the number of kills by helicopters will then be printed out.

FIGURE 22

MAIN PROCEDURE FOR PROCEDURE AIR




reset helicopter pointer
calculate no of missiles
call apport
call AD
call apport
pointer = nil?
N
Y
print loss matrix
print kill matrix
end

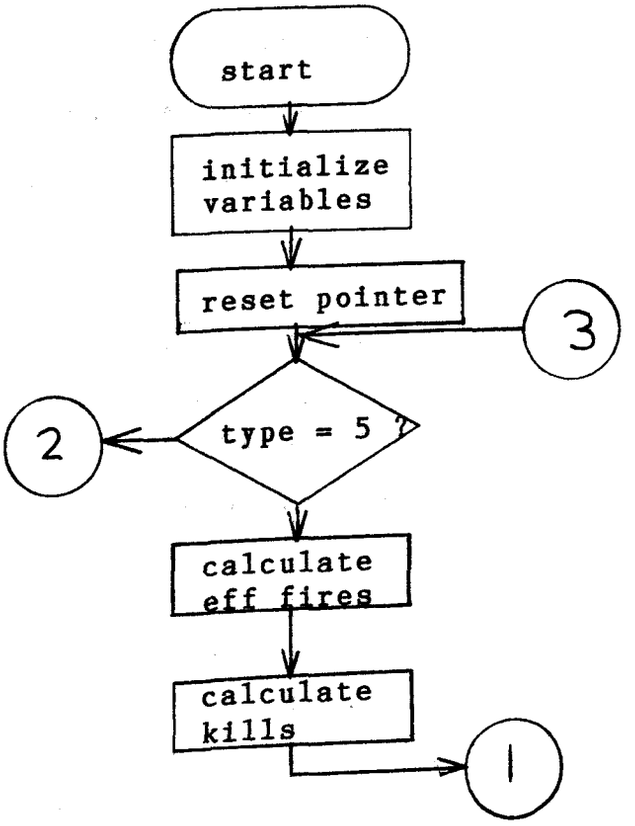
2.3.7 PROCEDURE AD (Air Defense). This procedure will allow air defense systems to kill attack helicopters. After variables are initialized a search is made for air defense weapons. For each weapon found the number of effective kills is found by the following formula:

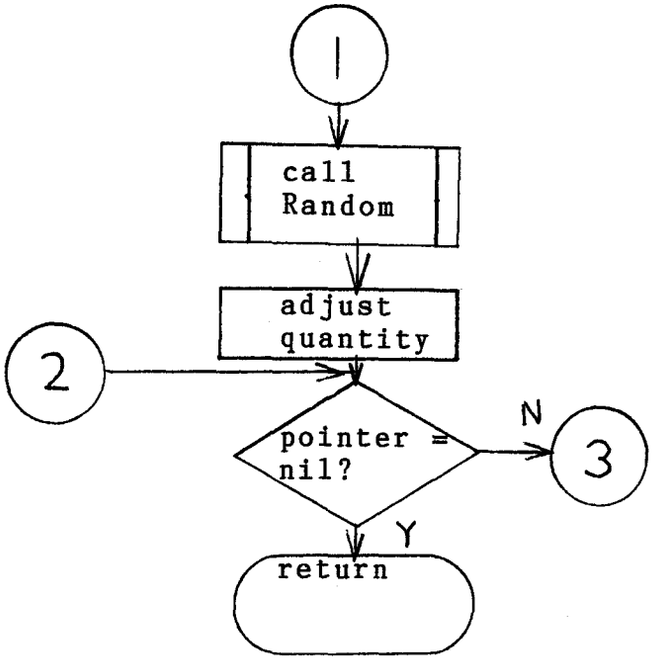
kills = quantity x probability(range) x probability of detection(range).

The number of downed aircraft is increased until end of the list is reached. The procedure will not allow more helicopters to be killed than originally existed.

FIGURE 23

PROCEDURE AD FOR PROCEDURE AIR





2.3.8. PROCEDURE ARMOR. The purpose of this procedure is to calculate losses due to direct fires.

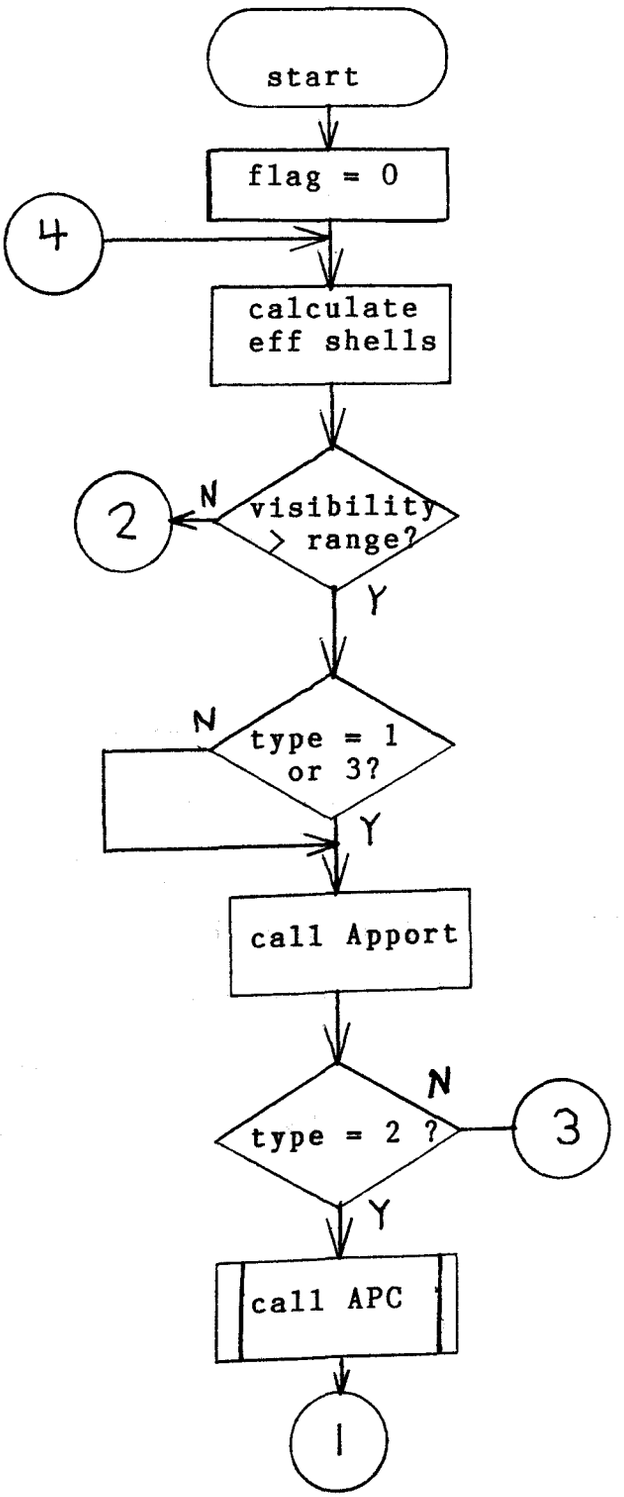
The list of weapons is printed out. The armor kill array is then initialized. The weapon pointer is reset to the beginning. The number of effective shells is computed using the following formula:

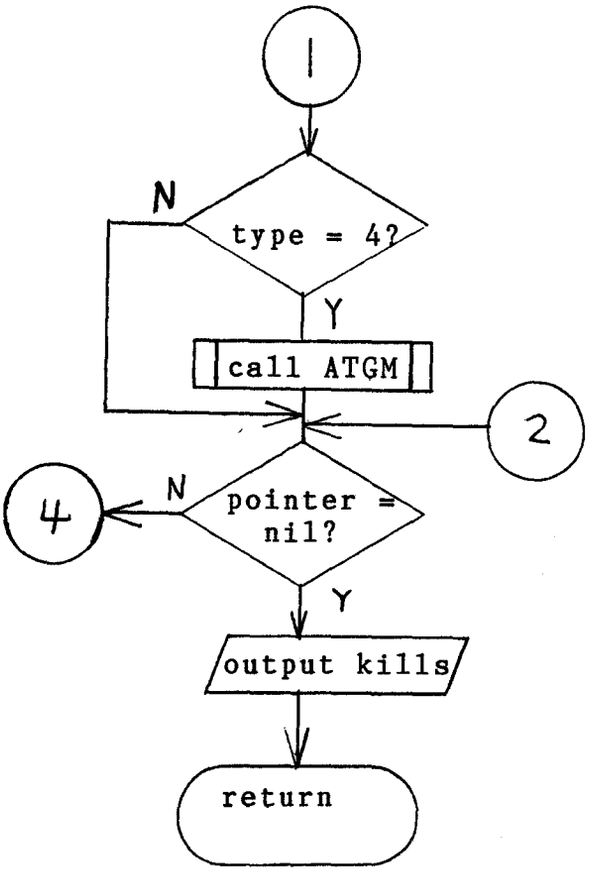
shells = no of weapons x the probability(range).

If the visibility is sufficient, Procedure Apport, ATGM, or APC will be called to assess losses.

FIGURE 24

MAIN PROCEDURE FOR PROCEDURE ARMOR

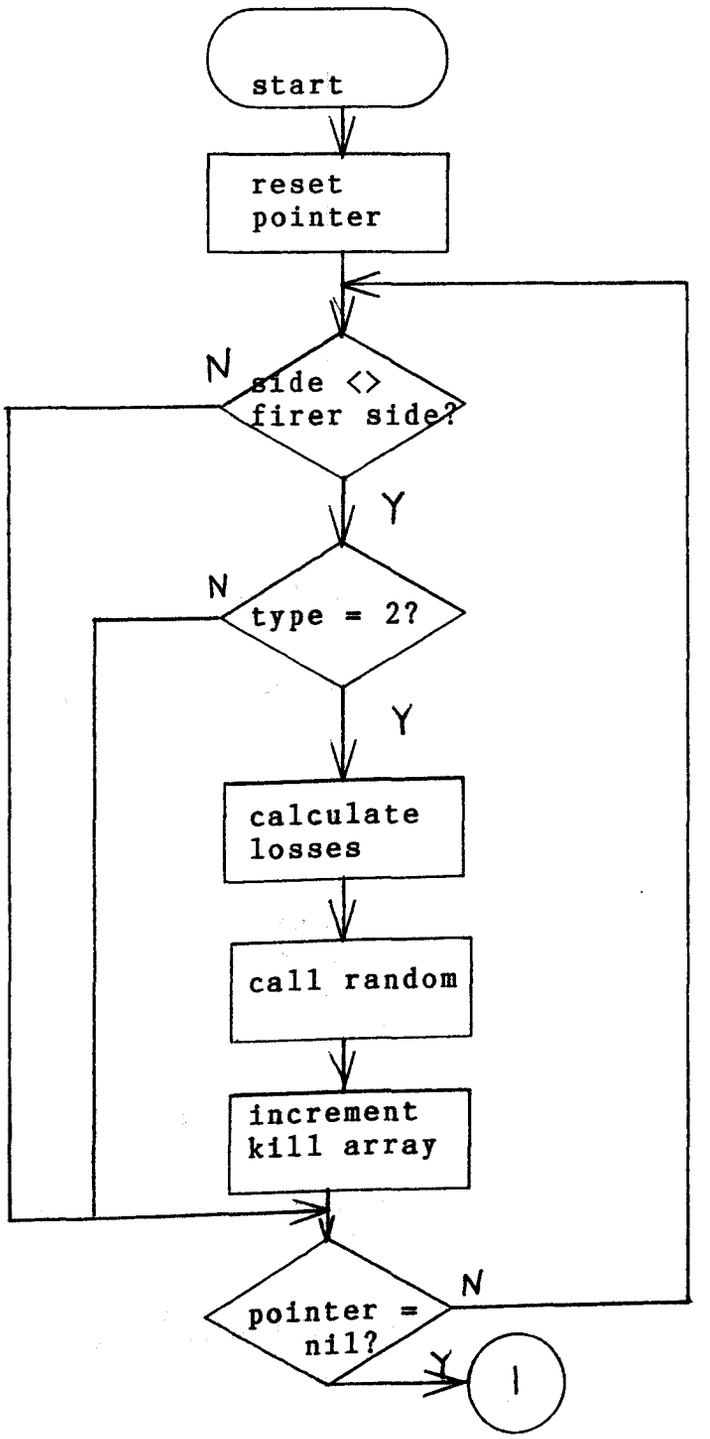


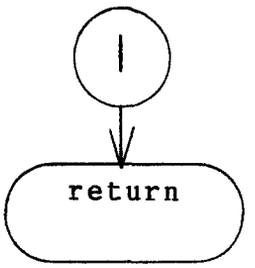


2.3.7 PROCEDURE APC. This procedure is used to allow Armored Personnel Carriers without an antitank capability to kill each other but not anything else. The weapon list is scanned and if the type of the target is 2, then NP kills will be assessed. Procedure random will then be called. As in all attrition routines there is a check to insure that more weapons than are loaded are not killed.

FIGURE 25

PROCEDURE APC FOR PROCEDURE ARMOR

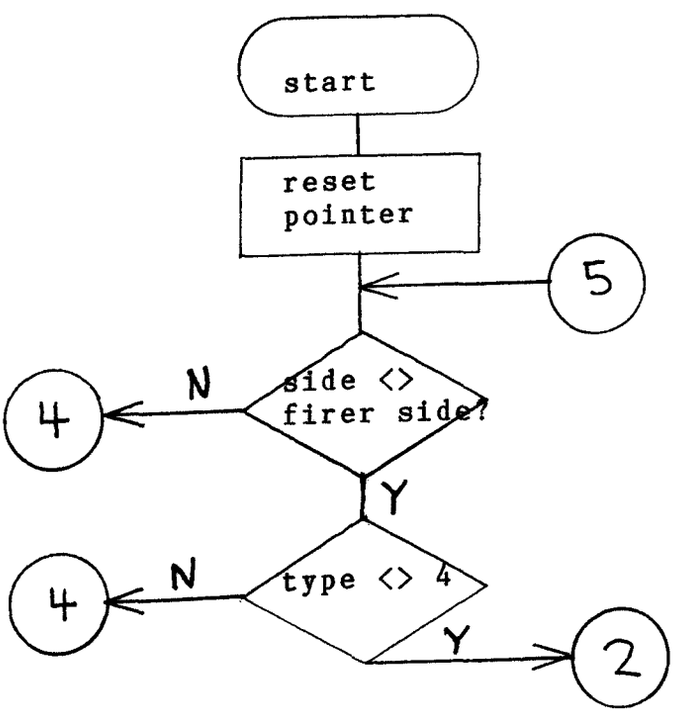


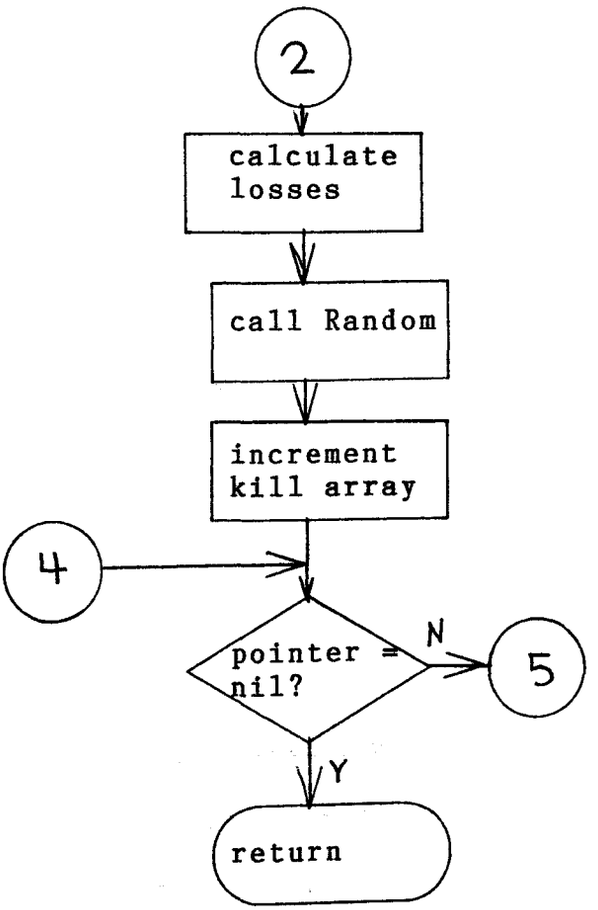


2.3.8 PROCEDURE ATGM. This procedure is used to allow antitank guided missiles to kill any armor system except another ATGM since it is highly unlikely that this would occur. The weapon list is scanned and if the type of the target is not 4, then Np kills will be assessed. Procedure Random will then be called. As in all assessment routines there is a check to insure that more weapons than are loaded are not killed.

FIGURE 26

PROCEDURE ATGM FOR PROCEDURE ARMOR




2
calculate losses
call Random
increment kill array
4
pointer = nil?
N
5
Y
return

CHAPTER III

CONCLUSION

The algorithms contained within the programs are not a panacea. There are numerous assumptions built in to these algorithms as described in chapter II. Since this is a low resolution model meant to quickly calculate attrition in a grossly simplified complex situation, there will always be room for changes. However, a note of caution is in order; there is a price to be paid for any "improvement" in the way of slower response time, more memory required, or unnecessary complexity for a low resolution model. Many " improvements" may, in effect, be "trade-offs".

3.1 ADOPTION TO STANDARD EQUIIPMENT. This game is designed for a "worst case" situation with a 128,000 Byte personal computer with one disk drive. At this point, the Department of Defense is buying thousands of personal computers which should allow the algorithms to take advantage of the memory available in these widely-distributed items. As an example, if these standard computers have 256,000 bytes, the extra memory would allow programs Prep and Game to be combined with plenty of room left over for additional features to be added.

3.2 AUTOMATIC UNIT ADJUSTMENT. It would be possible to develop an automatic unit adjusting routine that would decrement the number of weapons left after Program Game has been run. With a 128,000 Byte memory this may require the use of a fourth program. One of the disadvantages of this change

is the loss in flexibility in deciding which units should be decremented. A possible compromise would be to utilize a weighting factor which would allow the gamer to identify which units were the most likely to receive the highest levels of casualties.

3.3 SEPARATION OF ROUTINES. The attack helicopter and field artillery routines are so inherently complex that many rough approximations are necessary. Fragmentation into several different programs would allow considerable improvements to be made to those procedures. Problems would develop in integrating the data structures between the attrition programs and, needless to say, additional programs will considerably hamper the speed of the war gaming process and possibly nullify some of the advantages of an automatic war game.

3.4 INDEPENDENT GAMER DECISIONS. As noted in chapter II, both types of artillery are assumed to be firing at the same levels with the same percentage dedicated to counter battery fires. At the risk of further complicating the program additional coding could be utilized to allow red and blue gamers the options of altering firing levels and counterbattery fires.

3.5 ADDITIONAL IMPROVEMENTS. This simulation does not consider factors such as the use of smoke, logistics constraints, the use of night vision capabilities, guided projectiles, and the ability to automatically calculate losses to fixed-wing high performance aircraft. All of these and many

other factors could be included in the simulation with a corresponding loss in simplicity, response time, and free memory.

APPENDIX

PROGRAM LISTINGS

```
program build;

label 1;

{Define the records}

{Weapon characteristics record}

type weapon = record
     index: integer;
     kind: integer;
     name: string[10];
     prob: array[1..6] of real
     end;

{Unit record}

type unit = record
     name: string[15];
     now: integer;
     system: array[1..10,1..2] of integer;
     end;

{Artillery characteristics record}

type fa = record
     piece: string[10];
     kode: integer;
     rate: integer;
     size: integer;
     end;

{Artillery unit record}

type funit = record
     nam: string[15];
     tube: integer;
     siz: integer;
     end;
```

```
{Attack helicopter record}

type ah = record
     chop: string[15];
     tipe: integer;
     rang: integer;
     side: boolean;
     sort: integer;
     prob: real
     end;

{Artillery effects record}

type mat = record
     burst : array[1..5,1..4] of real;
     end;

{List user defined types}

type weaponfile = file of weapon;
type unitfile = file of unit;
type numfile = file of integer;
type funitfile = file of funit;
type fafile = file of fa;
type nunitfile = file of integer;
type ahfile = file of ah;
type burstfile = file of mat;

{List all Global variables}

var at: ah;
var bur: mat;
var yy : real;
var f1: weaponfile;
var f2: burstfile;
var f3: unitfile;
var f4: numfile;
var f5: nunitfile;
var f6: fafile;
var f7: funitfile;
var f8: ahfile;
var weap: weapon;
var wcode,i,numwpns,nn,x,xx,j,cod,quant: integer;
var org: unit;
var choice: string[10];
var k,l,y: integer;
var v: real;
var fire: fa;
var bat: funit;
var dum: char;
```

```
{** Artillery Effectiveness Matrix **}

{This procedure allows the user to
input the effectiveness of Artillery
against the 5 general types of vehicles}


procedure matrix;

label 3;

begin

{Display the matrix to the gamer}

reset (f2);
read(f2,bur);

{Dunn-Kempf Circular Error Probables
(estimated) 81mm .07 105 .13 4.2 .28
122 .32 others .40 }

writeln('EFFECTIVENESS MATRIX');
writeln(' ');
writeln('TARGET TYPE');
writeln(' ');
writeln('         TANK  APC  APC-AT ATGM  AD');
writeln(' ');

for j := 1 to 4
    do begin
    case j of
       1: write('MORT ');
       2: write('LT   ');
       3: write('MED  ');
       4: write('HVY  ');
    end;

    for i := 1 to 5
        do begin
        write( bur.burst[i,j]:7:4);
        end;
        writeln(' ');
    end;

{If this portion is invoked the
Artillery effectiveness matrix will
be started from scratch}

writeln(' ');
writeln(' Do you want to redo the effectiveness');
writeln('effectiveness matrix ?');
```

```
writeln(' ');
read(kbd,dum);
if (dum <> 'y') then goto 3;
rewrite(f2);

{Input the effectivenesses}

for j := 1 to 4
    do begin
    case j of
       1: write('MORT ');
       2: write('LT   ');
       3: write('MED  ');
       4: write('HVY  ');
    end;

    for i := 1 to 5
        do begin
        write('enter effectiveness against ');
        case i of
           1: writeln('TANKS ');
           2: writeln('APC   ');
           3: writeln('APC WITH AT CAP.');
           4: writeln('ATGM');
           5: writeln('Air Defense');
        end;
        read(kbd,yy);
        bur.burst[i,j] := yy/100;
        writeln(' ');
        writeln( bur.burst[i,j]);
        writeln(' ');
        end;
    end;

{write the information to file 2}

write (f2,bur);
3: writeln(' ');

end;

{*** Attack helicopter-air defense ***}

{This procedure allows the gamer to
input the capabilities of the attack
helicopters}

procedure air;

procedure insert;
```

```
{This procedure will insert a new
record into the file}

begin

      { Input the name}

      writeln(' enter the name of the helicopter');
      read(kbd, at.chop);

      {Input the weapon code}

      writeln(' enter the weapon code');
      read(kbd,at.tipe);

      {Input the maximum range}

      writeln('enter the range of the weapon system');
      read(kbd,at.rang);

      {Enter red or blue}

      writeln('is it a blue aircraft?');
      writeln('enter y for yes');
      read(kbd,choice);
      if (choice = 'y')then at.side :=true
      else at.side := false;

      at.sort :=0;

      writeln('enter the probability');
      read(kbd,at.prob);

      {Write the helicopter records to
      file 8}

      write(f8,at);

end;


procedure build;

{This procedure will build a new file
from scratch}

begin

rewrite(f8);
writeln('add another helicopter');
writeln('enter y for yes');
read(kbd,dum);
```

```
while (dum = 'y') do begin
  insert;
  dum := 'n';
  end;
end;


procedure add;

{This procedure will add a new record
to the end of the file}

label 2;

begin

reset(f8);
seek(f8,filesize(f8));
2: writeln('add another weapon');
writeln('enter y for yes');
read(kbd,dum);
while (dum = 'y') do begin
insert;
dum := 'n';
goto 2;
end;
dum := 'n';

end;


Procedure update;

{This procedure will allow the gamer to
selectively review and modify every
record in the file}

begin

reset(f8);
i := 0;
while not eof(f8)
   do begin
   read(f8,at);
   i:= i+1;
   writeln(' ');
   writeln('   NAME    CODE     RANGE SIDE    PROB');
   writeln(' ');
   write(at.chop:7);
   write(at.tipe:6);
   write(at.rang:9);
   write(at.side:6);
```

```
   writeln(at.prob:7:2);
   writeln(' ');
   writeln('do you want to change this entry');
   writeln('enter y for yes');
   writeln(' ');
   read(kbd,dum);
   if (dum = 'y') then begin
     seek(f8,i-1);
     insert;
     end;
   end;

end;


{Main procedure. Select the
subprocedures to be invoked.}

begin

{Display the contents of the file}

writeln('   NAME   CODE    RANGE SIDE   PROB');
reset(f8);
while not eof(f8)
   do begin
   read(f8,at);
   write(at.chop:7);
   write(at.tipe:6);
   write(at.rang:9);
   write(at.side:6);
   writeln(at.prob:7:2);
   end;

writeln(' ');
writeln('select option');
writeln(' 1 create new table');
writeln(' 2 add another entry to end of table');
writeln(' 3- update entries in table');
writeln(' ');

read(kbd,choice);

{Check for proper input}

if (choice < '1' ) or (choice > '3')

then begin

{Print out error message}

   writeln('Wrong!!! enter a 1,2,or 3');
```

```
end

else begin

   case choice of
        '1' : build;
        '2' : add;
        '3' : update;
   end;

end;

end;

{******** WEAPONS UPDATE PROCEDURE *******}

{his procedure allows the gamer to
input the characteristics for direct fire
weapons}

Procedure wupdate;

Procedure insert;

{This procedure will insert a new
record into the file}

begin

writeln('insert weapon code');
read(kbd,wcode);
writeln(wcode);

{Enter the weapon type}

writeln('enter type of weapon');
writeln(' 1 - tank, 2- APC');
writeln(' 3 - APC with Antitank capability');
writeln(' 4 - ATGM, 5 - Air Defense');
read(kbd,xx);
writeln(' Type = ',xx);
weap.kind := xx;
weap.index := wcode;

{Enter the name}

writeln('insert name');
read(kbd,weap.name);
writeln(weap.name);

{Input the probabilities}
```

```
for j := 1 to 6
    do begin
    k:= j * 500;
    writeln('insert probability for',k,'m.');
    read(kbd,weap.prob[j]);
    writeln(weap.prob[j]);
    end;

{Write the record to file 1}

write(f1,weap);

end;

procedure build;

{This procedure will build a new file
from scratch}

begin

rewrite(f1);
writeln('add another weapon');
writeln('enter y for yes');
read(kbd,dum);
while (dum = 'y') do begin
insert;
dum := 'n';
end;

end;


procedure add;

{This procedure will add a new record
to the end of the file}

begin

dum := 'y';
reset(f1);
seek(f1,filesize(f1));
while (dum = 'y') do begin
insert;
writeln('add another weapon');
writeln('enter y for yes');
read(kbd,dum);
end;

end;
```

```
Procedure update;

{This procedure will allow the gamer to
selectively review and modify every
record in the file}

begin

reset(f1);

i := 0;
reset(f1);
while not eof(f1)
   do begin
   writeln('name  code  weapon type');
   read(f1,weap);
   i:= i+1;
   write(weap.name);
   write(weap.index:6);
   writeln(weap.kind:6);
   writeln('range  probability');
   for j:= 1 to 6
     do begin
     k := 500*j;
     writeln(k:6,weap.prob[j]:8:4);
     end;
   writeln(' ');
   writeln('do you want to change this entry');
   writeln('enter y for yes');
   read(kbd,dum);
   if (dum = 'y') then begin
     seek(f1,i-1);
     insert;
     end;
   end;

end;

{Main procedure. Select the
subprocedures to be invoked.}

begin

{Display the contents of the file}

reset(f1);
while not eof(f1)
   do begin
   writeln('name  code  weapon type');
   read(f1,weap);
   write(weap.name);
```

```
    write(weap.index:6);
    writeln(weap.kind:6);
    writeln('range  probability');
    for j:= 1 to 6
      do begin
      k := 500*j;
      writeln(k:6,weap.prob[j]:8:4);
      end;
    writeln(' ');
    end;

writeln('Select option');
writeln(' 1 create new table');
writeln(' 2 add another entry to end of table');
writeln(' 3- update entries in table');
writeln(' ');

read(kbd,choice);

{Check for proper input}

if (choice < '1' ) or (choice > '3')

then begin

{Print out error message}

   writeln('Wrong!!! enter a 1,2,or 3');

end

else begin

   case choice of
        '1' : build;
        '2' : add;
        '3' : update;
   end;

end;

end;
```

```
{******** UNIT UPDATE PROCEDURE ******}

Procedure uupdate;

{This procedure allows the gamer to
input the number and type of
weapons in each unit}

procedure insert;

{This procedure will insert a new
record into the file}

begin

{Input the name of the unit}

writeln('enter name of unit');
read(kbd,org.name);
writeln(org.name);

{Input the number of weapons
in the unit}

writeln('enter no of weapons in the unit');
read(kbd,x);
writeln(' There are',x,'weapons in',org.name);
org.now := x;

{Input the weapon code}

for j:= 1 to x
    do begin
    writeln('enter the weapons code');
    read(kbd,cod);
    org.system[j,1] := cod;

    {Input the number of that
    weapon type}

    writeln('enter the number of weapons');
    read(kbd,quant);
    org.system[j,2] := quant;
    end;

{Write the number to file 3}

write(f3,org);
end;
```

```
procedure add;

{This procedure will add a new record
to the end of the file}

begin

dum := 'y';
reset(f3);
seek(f3,filesize(f3));
while (dum = 'y') do begin
insert;
writeln('add another weapon');
writeln('enter y for yes');
read(kbd,dum);
end;

end;

Procedure update;

{This procedure will allow the gamer to
selectively review and modify every
record in the file}

begin

i := 0;
reset(f3);
while not eof(f3)
   do begin
   i:= i+1;
   read(f3,org);
   writeln(' ');
   writeln(org.name);
   writeln(' ');
   writeln('    code quantity');
   for k := 1 to org.now
       do begin
       for j := 1 to 2
           do begin
           write(org.system[k,j]:7);
           end;
       writeln(' ');
       end;
   writeln(' ');
   writeln('do you want to change this entry');
   writeln('enter y for yes');
   read(kbd,dum);
   if (dum = 'y') then begin
      seek(f3,i-1);
```

```
      writeln(' ');
      insert;
      end;
   end;

end;


procedure build;

{This procedure will build a new file
from scratch}

begin

rewrite(f3);
writeln('add another unit');
writeln('enter y for yes');
read(kbd,dum);
while (dum = 'y') do begin
  insert;
  dum := 'n';
  end;

end;

begin

{Display the contents of the file}

reset(f3);
while not eof(f3)
   do begin
   read(f3,org);
   writeln(org.name);
   writeln(' ');
   writeln('    code quantity');
   for k := 1 to org.now
       do begin
       for j := 1 to 2
           do begin
           write(org.system[k,j]:7);
           end;
       writeln(' ');
       end;
       writeln(' ');
   end;

{Main procedure. Select the
subprocedures to be invoked.}
```

```
writeln('Select option');
writeln(' 1 create new table');
writeln(' 2 add another entry to end of table');
writeln(' 3- update entries in table');
writeln(' ');

read(kbd,choice);

{Check for proper input}

if (choice < '1' ) or (choice > '3')

then begin

{Print out error message}

   writeln('Wrong!!! enter a 1,2,or 3');

end

else begin

   case choice of
        '1' : build;
        '2' : add;
        '3' : update;
   end;

end;

end;


{********  ARTILLERY PROCEDURE *******}

{This procedure allows the gamer to
enter Artillery characteristics and
load the artillery units}

Procedure arty;

procedure insert;

{This procedure will insert a new
record into the file}

begin

    {Input the name of the piece}

    writeln('enter name of weapon');
    read(kbd,fire.piece);

    {Input the weapon code}
```

```
    writeln('enter weapon code');
    read(kbd,fire.kode);

    {Input the maximum rate of fire}

    writeln('enter rate of fire');
    read(kbd,fire.rate);

    {Enter the size of the piece}

    writeln('enter size of tube');
    writeln('1-small mortar, 2-');
    writeln('light artillery of small');
    writeln('3-medium artillery, 4-');
    writeln('heavy artillery');
    read(kbd,fire.size);

    {Write the record to file 6}

    write(f6,fire);

end;

procedure build;

{This procedure will build a new file
from scratch}

begin

rewrite(f6);
writeln('add another weapon');
writeln('enter y for yes');
read(kbd,dum);
while (dum = 'y') do begin
insert;
dum := 'n';
writeln('add another weapon');
writeln('enter y for yes');
read(kbd,dum);
end;

end;

procedure add;

{This procedure will add a new record
to the end of the file}

begin
```

```
reset(f6);
seek(f6,filesize(f6));
writeln('add another weapon');
writeln('enter y for yes');
read(kbd,dum);
while (dum = 'y') do begin
insert;
dum := 'n';
writeln('add another weapon');
writeln('enter y for yes');
read(kbd,dum);
end;

end;

Procedure update;

{This procedure will allow the gamer to
selectively review and modify every
record in the file}

begin

reset(f6);
i := 0;
writeln('   name   code   max rate size  ');
writeln(' ');
reset(f6);
while not eof(f6)
   do begin
   i:= i+1;
   read(f6,fire);
   write(fire.piece:7);
   write(fire.kode:7);
   write(fire.rate:7);
   writeln(fire.size:7);
   writeln(' ');
   writeln('do you want to change this entry');
   writeln('enter y for yes');
   read(kbd,dum);
   if (dum = 'y') then begin
     seek(f6,i-1);
     writeln(' ');
     insert;
     end;
   end;

end;

{Main procedure. Select the
subprocedures to be invoked.}
```

```
begin

{Display the contents of the file}

writeln('   name   code   max rate qty.  ');
reset(f6);
while not eof(f6)
   do begin
   read(f6,fire);
   write(fire.piece:7);
   write(fire.kode:7);
   write(fire.rate:7);
   writeln(fire.size:7);
   end;

writeln(' ');
writeln('Select option');
writeln(' 1 create new table');
writeln(' 2 add another entry to end of table');
writeln(' 3- update entries in table');
writeln(' ');

read(kbd,choice);

{Check for proper input}

if (choice < '1' ) or (choice > '3')

then begin

{Print out error message}

   writeln('Wrong!!! enter a 1,2,or 3');

end

else begin

   case choice of
        '1' : build;
        '2' : add;
        '3' : update;
   end;

end;

end;

procedure artunit;

{This procedure will update the
artillery units}
```

```
procedure insert;

{This procedure will insert a new
record into the file}

begin

    {Input the name of the unit}

    writeln('enter name of unit');
    read(kbd,bat.nam);

    {Input the weapon code}

    writeln('enter weapon code');
    read(kbd,bat.tube);

    {Input the number of weapons
    in the Battery}

    writeln('enter Battery size');
    read(kbd,bat.siz);

    {Write the record to file 7}

    write(f7,bat);

end;

procedure build;

{This procedure will build a new file
from scratch}

begin

rewrite(f7);
writeln('add another Battery?');
writeln('enter y for yes');
read(kbd,dum);
while (dum = 'y') do begin
insert;
dum := 'n';
writeln('add another weapon');
writeln('enter y for yes');
read(kbd,dum);
end;

end;

procedure add;
```

```
{This procedure will add a new record
to the end of the file}

begin

reset(f7);
seek(f7,filesize(f7));
writeln('add another Battery?');
writeln('enter y for yes');
read(kbd,dum);
while (dum = 'y') do begin
insert;
dum := 'n';
writeln('add another Battery?');
writeln('enter y for yes');
read(kbd,dum);
end;

end;


Procedure update;

{This procedure will allow the gamer to
selectively review and modify every
record in the file}

begin

reset(f7);
i := 0;
writeln(' ');
writeln('   name   code    size  ');
writeln(' ');
reset(f7);
while not eof(f7)
   do begin
   i:= i+1;
   read(f7,bat);
   write(bat.nam:6);
   write(bat.tube:6);
   writeln(bat.siz:9);
   writeln(' ');
   writeln('do you want to change this entry');
   writeln('enter y for yes');
   writeln(' ');
   read(kbd,dum);
   if (dum = 'y') then begin
     seek(f7,i-1);
     insert;
     writeln(' ');
     end;
```

```
    end;

end;

begin

{Display the contents of the file}

reset(f7);

writeln(' name  code    quantity');
writeln(' ');
while not eof(f7)
   do begin
   read(f7,bat);
   write(bat.nam:6);
   write(bat.tube:6);
   writeln(bat.siz:9);
   end;

writeln(' ');
writeln('Select option');
writeln(' 1 create new table');
writeln(' 2 add another entry to end of table');
writeln(' 3- update entries in table');
writeln(' ');

read(kbd,choice);

{Check for proper input}

if (choice < '1' ) or (choice > '3')

then begin

{Print out error message}

   writeln('Wrong!!! enter a 1,2,or 3');

end

else begin

   case choice of
        '1' : build;
        '2' : add;
        '3' : update;
   end;

end;

end;
```

```
{********** MAIN PROGRAM ************}


begin

{ Reserve files and establish file
variables }

assign (f1,'weapon.dta');
assign (f2,'burst.dta');
assign (f3,'unit.dta');
assign (f4,'house.dta');
assign (f5,'num.dta');
assign (f6,'arty.dta');
assign (f7,'bat.dta');
assign (f8,'ah.dta');

{reset(f6);}

{Output the menu}


writeln('welcome to the force program');
writeln('What do you want to do?');
writeln(' ');
1: writeln(' 1-update weapons, 2-update units');
writeln(' 3- update artillery');
writeln(' 4- update artillery units');
writeln(' 5- update attack helicopters');
writeln(' 6-update artillery effectiveness');
writeln(' 7-quit');
writeln(' ');
read(kbd,choice);

{Check for proper input}

if (choice < '1' ) or (choice > '7')

then begin

{Print out error message}

   writeln('Wrong!!! enter a 1,2,or 3');
   goto 1;

end

else begin

   case choice of
        '1' : wupdate;
        '2' : uupdate;
```

```
          '3' : arty;
          '4' : artunit;
          '5' : air;
          '6' : matrix;
          '7' : writeln(' ');
    end;

end;

{Close all data files}

      close(f6);
      close(f2);
      close(f1);
      close(f3);
      close(f4);
      close(f5);
      close(f7);
      close(f8)

end.
```

```
program Prep;

{ ******** Definitions***************}

{ Record formats for data files
are defined here }

{ Weapon characteristics record}

type weapon = record
     index: integer;
     kind: integer;
     name: string[10];
     prob: array[1..6] of real
     end;

{ Unit record}

type unit = record
     name: string[15];
     now: integer;
     system: array[1..10,1..2] of integer;
     end;

{ Field Artillery record}

type fa = record
     piece: string[10];
     kode: integer;
     rate: integer;
     size: integer;
     end;

{ Field Artillery Battery record}

type funit = record
     nam: string[15];
     tube: integer;
     siz: integer;
     end;

{ Attack Helicopter record}

type ah = record
     chop: string[15];
     tipe: integer;
     rang: integer;
     side: boolean;
     sort: integer;
     prob: real;
     end;
```

```
{ Input parameter record}

type pdeg = record
     prep: string[10];
     px: real;
     terr: string[10];
     tx: real;
     sr:integer;
     plos: real;
     time: real;
     max: integer;
     end;

{ Helicopter linked list record}

type hel = ^heli;
     heli = record
          sort : integer;
          side: boolean;
          code: integer;
          rang: integer;
          name: string[15];
          prob: real;
          next: hel
          end;

{ Weapon and Artillery linked list
record}

type fors = ^force;
     force = record
          code: integer;
          qty : integer;
          dis: real;
          side: boolean;
          tipe: integer;
          rate: integer;
          name: string[15];
          prob: array[1..6] of real;
          next: fors
          end;

{Programmer defined types are listed
here}

type weaponfile = file of weapon;
type unitfile = file of unit;
type numfile = file of integer;
type nunitfile = file of integer;
type funitfile = file of funit;
type fafile = file of fa;
type ahfile = file of ah;
```

```
type pfile = file of pdeg;
type helfile = file of heli;
type forfile = file of force;
type fafors = ^force;
type faforfile = file of force;

{Global variables are listed here}

var chop: heli;
var port,x5,sup,pc,use : real;
var o,plus,flag1: integer;
var ij,jj,xx,p,11: integer;
var plos :real;
var yy : real;
var px,tx,deg: real;
var x2,x3:real;
var f1: weaponfile;
var f3: unitfile;
var f4: numfile;
var f5: nunitfile;
var f6: fafile;
var f7: funitfile;
var f8: ahfile;
var f9: pfile;
var f10 : forfile;
var f11 : faforfile;
var f12 : helfile;
var current,start,last: fors;
var c4 : integer;
var cannon,tank: force;
var weap: weapon;
var wcode,i,numwpns,nn,x,j,cod,m,n: integer;
var org: unit;
var choice,terr,prep: string[10];
var k,1: integer;
var rtotal,btotal,area,v,x1,iarea: real;
var bsum,rsum: real;
var bat: funit;
var fire: fa;
var at: ah;
var time,speed,max,sr,count: integer;
var flag:integer;
var side: boolean;
var mm,sort :integer;
var x6 : real;
var pd: pdeg;

{******* DEGRADATION **********}

{ This procedure accounts for factors
which limit the ability of firers to
engage their targets and writes the
```

```
record to the diskette}

procedure degrade;

begin

assign(f9,'pd.dta');

{ Define the degradation due to the
degree of preparation.}

case prep of
   'p': px := 3/10;
   'h': px := 7/10;
   'm': px := 1;
end;

{ Define the degredation due to the
degree of probability of having
line of sight to a target at a given
range. The average line of sight is
assumed to follow a normal distributiion
with a mean of 1500 m and a standard
deviation of 1000m }

xx := sr div 500;

case xx of
   0: plos :=  1       ;
   1: plos :=  84/100  ;
   2: plos :=  69/100  ;
   3: plos :=  5/10    ;
   4: plos :=  31/100  ;
   5: plos :=  16/100  ;
   6: plos :=  7/100   ;
end;

{ Define the degradation due to the
degree of preparation.}

case terr of
   'o': tx := 1;
   'r': tx := 75/100;
   'h': tx := 5/10;
end;

{Calculate total degradation}

deg:= px*plos*tx;

{Write the input parameters and
degradation factors to the
```

```
diskette}

rewrite(f9);
pd.prep :=prep;
pd.px := px;
pd.terr := terr;
pd.tx :=tx;
pd.sr := sr;
pd.plos := plos;
pd.time := time;
pd.max := max;
write(f9,pd);
close(f9);
end;

{********** Procedure Air ***********}

{This procedure will write the number
of sorties to be flown by each aircraft
to the diskette}

Procedure air;

begin

{Display the contents of the file
and select the number of sorties
for each helicopter }

assign(f12,'hel.dta');
rewrite(f12);
reset(f8);
while not eof(f8)
   do begin
   read(f8,at);
   write('Name  ',at.chop);
   writeln('weapon code ',at.tipe);
   write('Range ',at.rang);
   if(at.side) then writeln('Blue')
   else writeln('Red');

   { Input the number of sorties}

   writeln(' ');
   writeln('Enter number of sorties');
   read(kbd,k);
   writeln(' ');

   { Write the information to the
   linked list file}

   chop.name := at.chop;
```

```
    chop.rang := at.rang;
    chop.code := at.tipe;
    chop.sort := k;
    chop.side := at.side;
    chop.prob := at.prob;
    write(f12,chop);
    end;

close(f12);

end;

{******* Procedure Menu **********}

{ This procedure allows the gamer to
input specific parameters into the
game}

Procedure menu;

begin

{ Enter the length of time of the game}

writeln('enter the length of time in minutes'); read(kbd,time);
writeln(' ');
writeln('the length of time is',time,'minutes');
writeln(' ');

{ Input the general type of terrain}

writeln('enter the type of terrain');
writeln(' ');
writeln('o for open,r for rolling,h for hilly');
read(kbd,terr);
writeln(' ');
writeln(terr);
writeln(' ');

{ Enter the distance between the
opposing forces}

writeln('enter the slant range ');
read(kbd,sr);
writeln(' ');
writeln(sr);
writeln(' ');

{Enter the maximum visibility}

writeln('enter the max visibility in m.');
read(kbd,max);
```

```
writeln(' ');
writeln(max);
writeln(' ');

{Designate the degree of preparedness
of the defensive positions}

writeln('enter the preperation factor');
writeln('enter p for prepared defenses');
writeln('enter h for hasty defenses');
writeln('enter m for meeting engagement');
read(kbd,prep);
writeln(' ');
writeln(prep);
writeln(' ');

{ Call Procedure Degrade and write
the input file to the diskette}

degrade;

end;
```

```
{***** Procedure Startup ***********}

{This procedure will initialize the
data structures and printout initial
lists of equipment for the gamer}

Procedure startup;

begin

{ Initialize the variables }

btotal := 0;
rtotal :=0;
rsum   :=0;
bsum   :=0;

{ Prepare input files}

reset(f3);
reset(f4);
reset(f5);
reset(f1);

{ Build the weapon linked list}

start := nil;
while not eof(f1)
      do begin
      read(f1,weap);
      new(current);
      current^.code := weap.index;
      current^.qty := 0;
      if(weap.index < 50) then
      current^.side := true
      else current^.side := false;
      current^.tipe := weap.kind;
      current^.rate := 0;
      current^.name := weap.name;
      for i := 1 to 6
          do begin
          current^.prob[i] := weap.prob[i];
          end;
      current^.next := start;
      start := current;
      end;

read(f4,numwpns);
read(f5,nn);

{ Print heading information }
```

```
writeln('UNITS LOADED');
writeln(' ');

while not eof(f3)
   do begin
   read(f3,org);
   writeln(org.name);

      { Add up the number of each
      type of weapon and load that
      information into the weapon
      linked list }

      for j:= 1 to org.now
      do begin
         k := org.system[j,1];
         1 := org.system[j,2];
         current := start;
         while(current^.code <> k)
         do begin
            current := current^.next;
         end;
         current^.qty := current^.qty +1;

         {Calculate the sum of Red and
         Blue weapons}

         case current^.side of
            true : btotal := btotal +1;
            false : rtotal := rtotal +1;
         end;
      end;
   end;

writeln(' ');

{Calculate the initial distribution
factors and write that information
to the diskette}

rewrite(f10);
current := start;
while (current <> nil) do begin
   case current^.side of
      true:
      current^.dis := current^.qty/btotal;
      false:
      current^.dis := current^.qty/rtotal;
   end;
   with current^ do begin
      tank.code := code;
```

```
      tank.qty := qty;
      tank.dis := dis;
      tank.side := side;
      tank.rate := rate;
      tank.name := name;
      tank.tipe := tipe;
         for i := 1 to 6
         do begin
            tank.prob[i] :=prob[i];
         end;
      end;
   write(f10,tank);
   current := current^.next;
end;

close(f10);

{ Build the field artillery linked
list }

assign(f11,'fafor.dta');
start:= nil;
rewrite(f11);
reset(f6);

{Read the information from the
field artillery unit file and
enter it into the linked list. }

writeln('ARTILLERY UNITS LOADED');
writeln(' ');

while not eof(f6)
do begin
   read(f6,fire);
   new(current);
   current^.code := fire.kode;
   current^.qty := 0;
   if(fire.kode < 11) then
   current^.side := true
   else current^.side := false;
   current^.tipe := fire.size;
   current^.name := fire.piece;
   current^.rate := fire.rate;
   current^.next := start;
   start := current;
end;

{ Calculate the quantity of each type
of weapon and enter it into the linked
list}
```

```
reset(f7);
while not eof(f7)
do begin
   read(f7,bat);
   writeln(bat.nam);
   current := start;
   while(current^.code <> bat.tube)
   do begin
      current := current^.next;
   end;
   l := bat.siz;
   current^.qty := current^.qty + 1;
   case current^.side of
      true: bsum := bsum + 1;
      false : rsum := rsum +1;
   end;
end;
writeln(' ');

{Calculate the distribution factors and
write the complete list to the diskette}

current:= start;
while (current <> nil) do begin
   case current^.side of
      true:
      current^.dis := current^.qty/btotal;
      false:
      current^.dis := current^.qty/rtotal;
   end;
   with current^ do begin
      cannon.code := code;
      cannon.qty := qty;
      cannon.dis := dis;
      cannon.side := side;
      cannon.tipe := tipe;
      cannon.rate := rate;
      cannon.name:= name;
   end;
   write(f11,cannon);
   current := current^.next;
end;

close(f11);

end;

{********** MAIN PROGRAM ************}

begin
```

```
{ Reserve files and establish file variables }

assign (f1,'weapon.dta');
assign (f3,'unit.dta');
assign (f4,'house.dta');
assign (f5,'num.dta');
assign (f6,'arty.dta');
assign (f7,'bat.dta');
assign (f8,'ah.dta');
assign(f10,'force.dta');

{ Build the Artillery and Weapon
linked lists}

startup;

{Input the gamer choices}

menu;

{Build the Attack Helicopter linked
lists}

air;

{ close all data files}

close(f1);
close(f3);
close(f4);
close(f5);
close(f6);
close(f7);
close(f8)

end.
```

```
program game;

{ ******** Definitions***************}

{ Record formats for data files
are defined here }

label 1;

{Unit record}

type unit = record
     name: string[15];
     now: integer;
     system: array[1..10,1..2] of integer;
     end;

{Field Artillery Battery record}

type funit = record
     nam: string[15];
     tube: integer;
     siz: integer;
     end;

{Field Artillery effectiveness Matrix}

type mat = record
     burst : array[1..5,1..4] of real;
     end;

{Helicopter linked list record}

type hel = ^heli;
     heli = record
            sort : integer;
            side: boolean;
            code: integer;
            rang: integer;
            name: string[15];
            prob: real;
            next: hel
            end;

{Input parameter list}

type pdeg = record
     prep: string[10];
     px: real;
     terr: string[10];
     tx: real;
     sr: integer;
```

```
      plos: real;
      time: real;
      max: integer;
      end;

{Weapon characteristics record}

type fors = ^force;
      force = record
            code: integer;
            qty : integer;
            dis: real;
            side: boolean;
            tipe: integer;
            rate: integer;
            name: string[15];
            prob: array[1..6] of real;
            next: fors
            end;

{programmer defined types are listed here}

type tubeloss = array[1..100] of integer;
type kill = array[1..100] of integer;
type unitfile = file of unit;
type numfile = file of integer;
type nunitfile = file of integer;
type funitfile = file of funit;
type burstfile = file of mat;
type quant = array[1..100] of integer;
type helfile = file of heli;
type pdegfile = file of pdeg;
type forfile = file of force;
type can = ^force;
type fafors = can;
type faforfile = file of force;

{Global variables are listed here}

var port,x5,x6,pc,use : real;
var tub,itub : tubeloss;
var quan,down: quant;
var y,adf: real;
var pdec,ahf: real;
var ll: integer;
var plos :real;
var shell: real;
var bur: mat;
var px,tx,deg: real;
var eff: real;
var artkil,armkil,helkil,init,kil: kill;
var x2,x3:real;
```

```
var f2: burstfile;
var f3: unitfile;
var f4: numfile;
var f5: nunitfile;
var f7: funitfile;
var wcode,i,numwpns,nn,x,j,cod,m,n: integer;
var org: unit;
var ans,choice,terr,prep: string[10];
var k,l: integer;
var rtotal,btotal,area,v,xl,iarea: real;
var bsum,rsum: real;
var bat: funit;
var how: can;
var time,speed,max,sr,count: integer;
var flag:integer;
var side: boolean;
var f9 : pdegfile;
var f12 : helfile;
var this,star,las,place: hel;
var chop : heli;
var pd: pdeg;
var f11 : faforfile;
var start,here,that :fors;
var beg,current,curr : can;
var f10 : forfile;
var cannon, tank : force;

{*******  Random Procedure **********}

{The purpose of this procedure is to
use a random number generator to
determine whether or not an extra loss
should be added due to the fractional
portion of the loss generating
procedure. }

procedure random(var extra:real);

begin

{ Augment the  number of kills if
the random number is smaller than
the fractional portion of the loss
generating procedure}

if((random) <extra ) then
   ll := ll +l;

end;

{******* Distribution Procedure *****}
```

```
{This procedure will recalculate the
distribution factors for each system
loaded in the game. This game assumes
that each target is equally likely to
be engaged by any given firer}

procedure distribute;

begin

rtotal := 0;
btotal := 0;

{Compute new totals }

current := start;
while (current <> nil) do begin
   case side of
      true: btotal := btotal + current^.qty;
      false: rtotal := rtotal + current^.qty;
   end;

   current:= current^.next;
end;

{Recompute the new distribution
factors}

while (current <> nil) do begin
   if (btotal >0 ) and (rtotal > 0)
   then begin
      if (current^.side) then begin
         if (btotal <= 0) then current^.dis :=0
         else
            current^.dis := current^.qty/btotal;
         end
         else begin
            if (rtotal <= 0) then current^.dis := 0
            else
            current^.dis := current^.qty/rtotal;
         end;
   end;

   current := current^.next;
end;

end;

{*******  Apportionment Procedure****}

procedure apport(var loss:real);
```

```
{ This procedure accepts a total of
successful fires from another procedure
and apportions the losses to each
particular weapon type }

{ASSESS LOSSES}

begin

ll := 0;

here := start;

while( here <> nil) do begin
   if(side <> here^.side) then begin


   {Calculate the probablility of
   a firer being able to detect and
   range upon a target. Assume a
   linear relation between range and
   the probability. }

   pdec := 1-(pd.sr/5000);

   {Degrade the effects of the weapon
   systems. Only Blue gets the
   advantage of defensive preperations}

   case here^.side of
      true: deg := pd.px*pd.tx*pd.plos;
      false: deg := pd.tx*pd.plos;
   end;
      x2 := here^.dis *loss*deg*pdec;

      {Apply range and error factors
      for artillery fires}

      if(flag > 0) then begin
         x2 := x2 * bur.burst[flag,here^.tipe];
         x2 := x2/(pdec*pd.plos);
         end;
         ll := trunc(x2);
         x3 := frac(x2);
         random(x3);
         i := here^.code;

         { Carry over previous kills
         to that weapon type }

        kil[i] := kil[i] +ll;
     end;
```

```
    { Do not allow the program
    to kill more weapons than
    are loaded }

    if (kil[i] >=init[i] ) then
    kil[i] := init[i];

    here := here^.next;

end;

flag := 0;

end;

{***** INDIRECT FIRE PROCEDURE ******}

procedure artillery;

label 2;

{ Print out the artillery array }

procedure print;

begin

writeln('ARTILLERY REMAINING ');
writeln(' ');
writeln('    WEAPON      QTY ');
writeln(' ');
current := beg;

while( current <> nil )
do begin
   writeln(current^.code:10,current^.qty:4);
   current := current^.next;
end;
writeln(' ');

end;

{Calculate losses to artillery pieces}

procedure counter(var xxx: real);

begin

{Calculate the number of tubes
destroyed }

11 := 0;
```

```
current := beg;

while(current^.next <> nil)
do begin
   if(side <> current^.side) then begin
      flag := curr^.tipe;
      port := bur.burst[flag,4]*xxx;
      port := port* current^.dis;
      11 := trunc(port);
      x3 := frac(port);
      random(x3);
      i := curr^.code;
      tub[i] := tub[i] + 11;

      {Do not kill more than the number
      of tubes originally loaded}

      if(tub[i] > itub[i]) then begin
         tub[i] := itub[i];
         end;
      end;

   current := current^.next;
   end;

end;

{Main Field Artillery Procedure}

begin

writeln('FIELD ARTILLERY ROUTINE ');
writeln(' ');

{Clear Artillery loss array}

for i := 1 to 100
    do begin
    tub[i] := 0;
    end;

{Initiate the artkil array}

for i := 1 to 100
    do begin
    artkil[i] := kil[i]
    end;

{Read the effectiveness matrix}

reset(f2);
read(f2,bur);
```

```
{Select the ammunition usage rate}

writeln('Select Firing rates');
2: writeln(' 1-low, 2-medium');
writeln(' 3-high,4-very high ');
read(kbd,ans);
writeln(' ');

{Select the percentage of fires
to be used against enemy artillery}

writeln('Select percentage for counter');
writeln('Battery fire');
read(kbd,pc);
writeln(' ');

{Check for proper input}

if (choice < '1' ) or (choice > '4')

then begin

{Print out error message}

   writeln('Wrong!!! enter a 1,2,3 or 4');
   goto 2;

end

else begin

{Calculate the rate of ammunition usage}

   case ans of
      '1' : use := 1/48;
      '2' : use := 1/24;
      '3' : use := 1/12;
      '4' : use := 1/6;
   end;

end;



{Print out artillery array}

print;

current := beg;
while (current <> nil)
   do begin
```

```
    {Calculate total shellfire}

    iarea:= current^.rate*(pd.time/60)*use*(1.-pc);
    eff := iarea * current^.qty;
    flag := current^.tipe;
    side := current^.side;
    curr := current;
    y := current^.rate*(pd.time/60)*use*pc;
    y := y* current^.qty;
    if(pc > 0) then
       counter(y);
    current := curr;
    m := current^.code;
    if (tub[m] <>0) then begin
       if(tub[m] < current^.qty)
          then current^.qty := current^.qty -tub[m]
          else current^.qty := 0;
       end;

    {Apportion Artillery kills}

    iarea:= current^.rate*(pd.time/60)*use*(1.-pc);
    eff := iarea * current^.qty;
    apport(eff);
    flag := 0;
    current := current^.next;
end;

close(f11);

{ Print out the list of artillery tubes
killed}

writeln('ARTILLERY TUBES DESTROYED');
writeln(' ');

writeln('    WEAPON      QTY ');
writeln(' ');

for i := 1 to 100
    do begin
    if(tub[i] <>0) then
       writeln(i:10,tub[i]:4);
end;


writeln(' ');

{ Print out list of artillery }

print;
```

```
{Print out the list of weapons
destroyed by artillery}

writeln(' ');
writeln('ARTILERY KILLS');
writeln(' ');
{writeln(lst,'ARTILERY KILLS');
writeln(lst,'WEAPON CODE  QUANTITY');
}
for i := 1 to 100
    do begin
    artkil[i] := kil[i] - artkil[i];
    if(artkil[i] <>0) then
       writeln(i:10,artkil[i]:4);
end;
writeln(' ');
end;
```

```
{*** Attack Helicopter/ ADA routine ***}

{ This procedure will calculate the
attrition due to the introduction of attack
helicopters}

Procedure air;

{This procedure will calculate aircraft
losses due to air defense}

Procedure ad;

begin

here := start;

{ Initialize variables}

1 := round(pd.sr/500);
adf:= 0;
x5 := 0;
x6 := 0;

{ Calculate the number of effective
air defense fires}

while( here <> nil)
   do begin

   {Check for air defense types}

   if(here^.tipe = 5) then begin
      if(side <> here^.side)
      then begin .
         adf := here^.qty * here^.prob[1];
         x5 := adf * pdec;
         adf := x5;
         down[k] := trunc(adf) + down[k];
         x6 := frac(adf);
         random(x6);
         down[k] := down[k] + 11;
         11 := 0;
         if(down[k] > quan[k]) then
         down[k] := quan[k];
      end;
   end;
   here := here^.next;
end;

end;
```

```
{MAIN ATTACK HELICOPTER ROUTINE}

begin

writeln('ATTACK HELICOPTER PROCEDURE');
writeln(' ');

{Initiate the helkil array}

for i := 1 to 100
    do begin
    helkil[i] := kil[i]
    end;

{Display the air defense systems }

writeln('AIR DEFENSE SYSTEMS');
writeln('    WEAPON      QTY ');
writeln(' ');
here := start;

while(here <> nil)
     do begin
     if(here^.tipe = 5)
     then writeln(here^.name:10,here^.qty:8);
     here := here^.next;
     end;
writeln(' ');

{build the helicopter linked list}

{ Clear the helicopter
loss matrix }

for j := 1 to 100
do begin
   quan[j] := 0;
   down[j] := 0;
end;

assign(f12,'hel.dta');

star := nil;

reset(f12);
while not eof(f12)
do begin
   new(this);
   read(f12,chop);
   with this^ do begin
   name := chop.name;
   rang := chop.rang;
```

```
    code := chop.code;
    sort := chop.sort;
    side := chop.side;
    prob := chop.prob;
    j := code;
    quan[j] := sort;
    end;

this^.next := star;
   star:= this;
end;

close(f12);

{Initialize variables}

ahf:= 0;
adf:= 0;
flag:=0;

this := star;

{Calculate the number of effective
missiles fired}

writeln('HELICOPTERS LOADED');
writeln(' ');

while(this <> nil)
do begin
   ahf := this^.sort*4*this^.prob;
   side := this^.side;

   {check for the maximum range}

   if (pd.sr < this^.rang) and
   (this^.sort <>0)
   then begin
      place := this;
      k:= this^.code;
      writeln(this^.name);
      ahf := ahf/2;
      apport(ahf);
      this := place;
      ad;
      this := place;
      this^.sort := quan[k] - down[k];
      ahf := this^.sort*2;
      apport (ahf);
      this := place ;
   end;
   this := this^.next;
```

```
end;
writeln(' ');

{Output the number of helicopters
killed}

writeln('HELICOPTERS KILLED');
writeln(' ');
writeln('    WEAPON      QTY ');
writeln(' ');
for j := 1 to 100
do begin
   if(down[j] <> 0) then
   writeln(j:10,down[j]:8);
end;
writeln(' ');

{Output the number of helicopter
kills}

writeln('ATTACK HELICOPTER KILLS');
writeln(' ');

for i := 1 to 100
    do begin
    helkil[i] := kil[i] - helkil[i];
    if(helkil[i] <>0) then
       writeln(i:10,helkil[i]:4);
end;
writeln(' ');
end;

{***** Direct Fire Routine **********}

Procedure armor;

{ASSESS APC KILLS}

{This procedure is necessary because
APC's can be killed by all other types
of weapons but they can only kill another
APC}

procedure apc (var loss: real);

begin

11 := 0;

here := start;


while (here <> nil)
```

```
while (here <> nil)
do begin
   if (side <> here^.side) then begin
      if (here^.tipe = 2) then begin
         x2 := here^.dis *loss*deg;
         ll := trunc(x2);
         x3 := frac(x2);
         random(x3);
         i := here^.code;
         kil[i] := kil[i] +ll;
      end;

     if (kil[i] >= init[i] ) then
     kil[i] := init[i];
   end;
   here := here^.next;
end;

end;

{ASSESS ATGM KILLS}

{This Procedure is necessary because all
other weapon types can kill an ATGM except
another ATGM}

procedure atgm(var loss: real);

begin

ll := 0;

here := start;

while (here <> nil)
do begin
   if (side <> here^.side) then begin

   {Check for ATGM type}

      if (here^.tipe <> 4) then begin
         x2 := here^.dis *loss*deg;
         ll := trunc(x2);
         x3 := frac(x2);
         random(x3);
         i := here^.code;
         kil[i] := kil[i] +ll;
      end;

      {Do not allow the game to kill
      more items than are loaded}
```

```
      if (kil[i] >=init[i] ) then
      kil[i] := init[i];
   end;
   here := here^.next;
end;

end;

{****** MAIN ARMOR PROCEDURE *********}

{This procedure conducts all types of
armor engagements and calls Procedures
APC and ATGM when applicable }

begin

{ Print Heading Information}

writeln('ARMOR PROCEDURE');
writeln(' ');

{Output the list of weapons }

writeln('The following weapons are loaded');
writeln('into the game');
writeln('    WEAPON      QTY ');
writeln(' ');
here := start;
while ( here <> nil)
do begin
   if(here^.qty <> 0) then begin
      writeln(here^.name:10,'    ',here^.qty:4);
   end;
here := here^.next;
end;

{Initiate the armkil array}

for i := 1 to 100
    do begin
    armkil[i] := kil[i]
    end;

{Initialize variables}

here := start;
flag := 0;

while(here <> nil)
   do begin
```

```
   {calculate the number of
   effective engagements}

   l := round(pd.sr/500);
   shell := here^.qty * here^.prob[l];

   { Check for sufficient visibility}

   if(pd.sr > pd.max) then shell := 0;

   that := here;
   flag :=0;
   side := here^.side;

   {Apportion losses}
   if (shell > 0) then begin
      case here^.tipe of
         1 : apport(shell);
         2 : apc(shell);
         3 : apport(shell);
         4 : atgm(shell);
         5 : write(' ');
      end;
   end;
   here := that;
   here := here^.next;
end;
writeln(' ');

{Output Armor Kills}

writeln('ARMOR KILLS');
writeln(' ');
writeln('WEAPON   QTY');

for i := 1 to 100
    do begin
    armkil[i] := kil[i] - armkil[i];
    if(armkil[i] <>0) then
       writeln(i:10,armkil[i]:4);
end;
writeln(' ');

end;

{********** MAIN PROGRAM ************}

begin

{ Reserve files and establish file
variables }
```

```
assign (f2,'burst.dta');
assign (f3,'unit.dta');
assign (f4,'house.dta');
assign (f5,'num.dta');
assign (f7,'bat.dta');
assign (f10,'force.dta');
assign (f9,'pd.dta');

writeln('WELCOME TO THE WARGAME');

{Read the list of input parameters
from the Prep program}

reset(f9);
read(f9,pd);

{Calculate the degredation factors}

deg := pd.px*pd.tx*pd.plos;

close(f9);

{Clear the kill matrixes,
the initial matrix and
the initial tube matrix }

for i := 1 to 100
   do begin
   kil[i] := 0;
   helkil[i] := 0;
   artkil[i] := 0;
   armkil[i] := 0;
   init[i] := 0;
   itub[i] := 0;
   end;

{ Load the forces from the data base}

start := nil;

reset(f10);
while not eof(f10)
do begin
   new(here);
   read(f10,tank);
   with here^ do begin
      code :=tank.code;
      qty :=tank.qty;
      dis:= tank.dis;
      side := tank.side;
      rate := tank.rate;
```

```
      name := tank.name;
      tipe := tank.tipe;
   end;
   for i := 1 to 6
   do begin
      here^.prob[i] := tank.prob[i];
   end;
   j := here^.code;
   init[j] := here^.qty;
   here^.next := start;
   start := here;
end;

close(f10);

assign (f11,'fafor.dta');
reset(f11);

{load artillery piece array}

beg := nil;
while not eof(f11)
   do begin
   read(f11,cannon);
   new(current);
   with current^
      do begin
      code := cannon.code;
      qty := cannon.qty;
      dis := cannon.dis;
      side := cannon.side;
      tipe := cannon.tipe;
      rate := cannon.rate;
      name := cannon.name;
   end;
   i := cannon.code;
   itub[i] := cannon.qty;
   current^.next := beg;
   beg:= current;
end;

close(f11);

{Output the list of input parameters}

writeln(' ');
writeln('INPUT VARIABLES ');
writeln(' ');
with pd
do begin
   writeln('Preparation factor ',prep );
   writeln('Terrain factor ',terr );
```

```
   writeln('Slant range ',sr );
   writeln('Time ',time:4:0 );
   writeln('Maximum visibility ',max );
end;
writeln(' ');
{Output the list of weapons }

writeln('The following weapons are loaded');
writeln('into the game');
writeln('    WEAPON      QTY ');
writeln(' ');
here := start;
while ( here <> nil)
do begin
   if(here^.qty <> 0) then begin
      writeln(here^.name:10,'    ',here^.qty:4);
   end;
here := here^.next;
end;

{Print out the list options for the
gamer}

writeln(' ');
1: writeln('What do you want to do?');
writeln(' 1- fight artillery');
writeln(' 2-fight attack helicopters');
writeln(' 3- conduct armor engagements');
writeln(' 4-quit');
writeln(' ');

{Input the choice}

read(kbd,choice);

{Check for proper input}

if (choice < '1' ) or (choice > '4')

then begin

{Print out error message}

   writeln('Wrong!!! enter a 1-4');
   goto 1;
end
else begin
   case choice of
      '1' : artillery;
      '2' : air;
      '3' : armor;
      '4' : writeln(' ');
```

```
    end;


 end;

 {Print out results}

 writeln('****** combat results****');
 writeln(' ');

 for i:= 1 to 100
    do begin
    if(init[i] <> 0) then begin
       writeln('code ',i:3,' initial ',init[i]:4,'
 killed',kil[i]:3);
       here := start;
       while (here^.code <> i) and (here <> nil)
       do begin
          here := here^.next;
       end;
       here^.qty := init[i] - kil[i];
       if( here^.qty < 0) then here^.qty := 0;
    end;
 end;
 writeln(' ');
 here := beg;

 {Recalculate the distribution factors}

 distribute;

 if(choice <> '4') then goto 1;

 { close all data files}

      close(f2);
      close(f3);
      close(f4);
      close(f5);
      close(f7)

end.
```

END NOTES

[1] US Department of the Army, Organizational Data for the Army in the Field (Ft Leavenworth, KS.: US Government Printing Office, May 1983), p. 5.

[2] US Department of the Army, Operational Terms and Symbols (Washington, D.C.: US Government Printing Office, October 1985), p. 2-5.

[3] US Department of the Army, Map Reading (Washington, D.C.: US Government Printing Ofice, January 1969), p. 6-1.

[4] US Department of the Army, Dunn-Kempf Combat Results Tables (Washington, D.C.: US Government Printing Office, n.d.), p. 5.

[5] Ibid., pp. 10-12.

[6] Ibid., pp. 1-6.

[7] Loren Radford and Roger W. Haigh, Turbo Pascal for the IBM PC (Boston: PWS Publishers, 1986), p. 317.

[8] LTC Wayne A. Silkett, " 72 Ways to Win Bigger," Infantry, September-October 1985, p. 38.

[9] Department of the Army, The Tank and Mechanized Infantry Team (Washington, D.C.: US Government Printing Office, June 1977), p. 5-6.

[10] Dunn-Kempf Combat Results Tables, p. 19.

[11] Irwin E. Miller and John E. Freund, Probability and Statistics for Engineers, (New Jersey: Prentice Hall, 1965), pp. 99-103.

[12] Dunn-Kempf Combat Results Tables, pp. 1-25.

BIBLIOGRAPHY

Miller, Irwin E., and Freund, John E.. Probability and Statistics for Engineers. New Jersey: Prentice Hall, 1965.

Radford, Loren E., and Haigh, Roger W.. Turbo Pascal for the IBM PC. Boston: PWS Publishers, 1986.

Silkett, LTC Wayne A.. "72 Ways to Win Bigger," Infantry, Sepember-October, 1985. pp. 36-40.

US Department of the Army. Dunn-Kempf Combat Results Tables. Washington, D.C.: US Government Printing Office, n.d..

Department of the Army. The Tank and Mechanized Infantry Team. Washington, D.C.: US Government Printing Office, June, 1977.

US Department of the Army. Organizational Data for the Army in the Field. Ft Leavenworth, KS.: US Government Printing Office, May, 1983.

US Department of the Army. Map Reading. Washington, D.C.: US Government Printing Office. January, 1969.

US Department of the Army. Operational Terms and Symbols. Washington,D.C.: US Government Printing Office, October, 1985.

Zohar, Nanna. "Special Relations in Automated Deduction," Journal of the Association of Computing Machinery 33 (January 1986): 1-59.

VITA

CHARLES EDMUND HARRISON III

Educational Background:

BS in Military Science with Area of Concentration in Applied Science and Engineering, United States Military Academy, West Point, NY.

Us Army Systems Engineering Analysis and Management School, Fort Bliss, TX., 1976.

US Army Command And General Staff College, 1983.

Military Background:

1972-74, Executive Officer, 82d Airborne Division, Ft Bragg, NC.

1974-1975 Executive Officer, 2d Infantry Division, Korea.

1976-1978, Assistant Operations Officer and Battery Comander, 2d Infantry Division, Korea.

1978-1980 War Gamer, Combined Arms Combat Development Agency, Ft Leavenworth, KS.

1980-1983, Battery Commander, Police Chief, 3d Infantry Division, Germany,

1983-1986, Assistant Professor of Military Science, TX A&I University, Kingsville, TX.